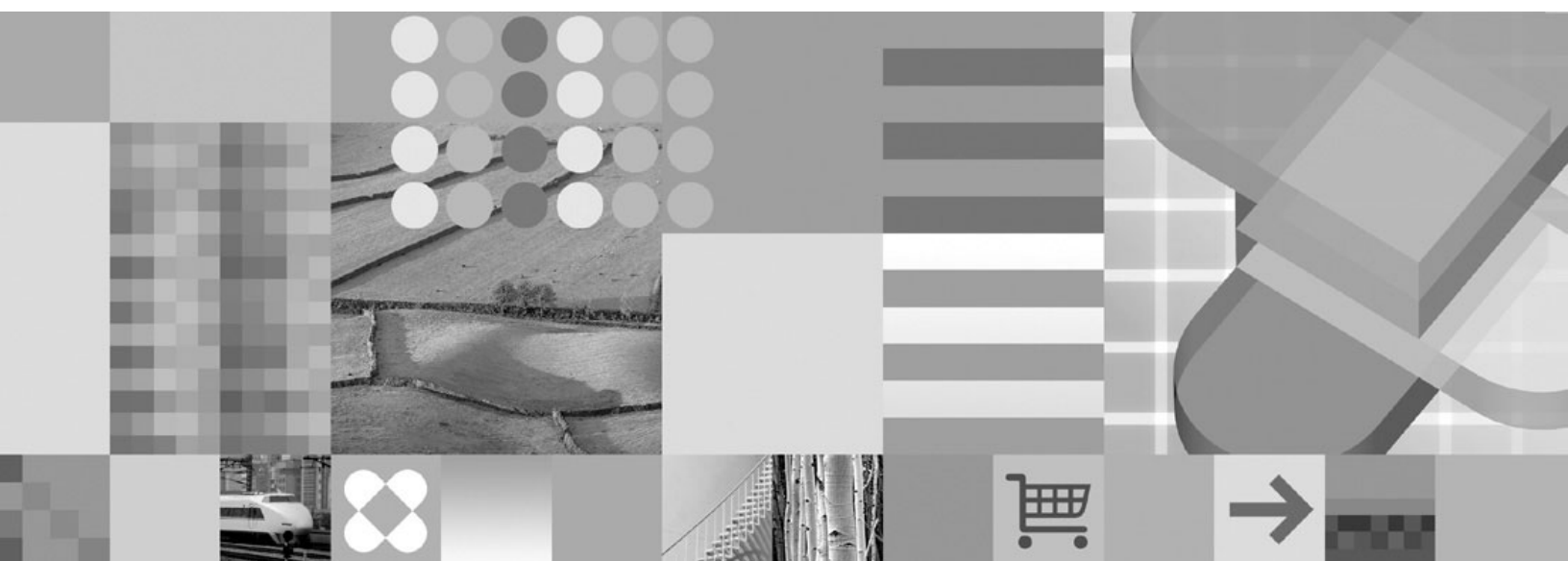




Application Programming Guide



Application Programming Guide

Note

Before using this information and the product it supports, read the information in "Notices" on page 567.

Fifth Edition (January 2005)

This edition applies to version 8, release 3 of IBM DB2 Content Manager Enterprise Edition (product number 5724-B19), version 8, release 3 of IBM DB2 Content Manager for z/OS (product number 5697-H60), and version 8, release 3 of IBM DB2 Information Integrator for Content (product number 5724-B19) and to all subsequent releases and modifications until otherwise indicated in new editions.

Copyright © 1990-2004 Captiva Software Corporation and/or its licensors, 10145 Pacific Heights Blvd., San Diego, CA 92121 U.S.A. All rights reserved.

Outside In® Image Export © 1992-2004 Stellent Chicago, Inc. All rights reserved.

Copyright © 2000 The Apache Software Foundation. All rights reserved. This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Redistribution and use in binary form, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the above disclaimer in the documentation and/or other materials provided with the distribution. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. The names "The Apache Logging Services Project" "log4j" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

Document Viewer, © 1991-2004 MS Technology, Inc. Charlotte, NC. All Rights Reserved.

IBM XSLT Processor

Licensed Materials - Property of IBM

© Copyright IBM Corp., 1999-2004. All Rights Reserved.

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

Copyright © 1998-2003 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)." The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)."

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

=====

This product includes cryptographic software written by Eric Young (ey@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Copyright © 1995-1998 Eric Young (ey@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (ey@cryptsoft.com). The implementation was written so as to conform with Netscape's SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. All advertising materials mentioning features or use of this software must display the following acknowledgement: "This product includes cryptographic software written by Eric Young (ey@cryptsoft.com)." The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)."

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION). HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

CUP Parser, © 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian, All Rights Reserved.

Permission to use, copy, modify, and distribute the CUP Parser Generator software and documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the names of the authors or their employers not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

The authors and their employers disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the authors or their employers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

This product includes software developed by the Eclipse Project (<http://www.eclipse.org/>).

© Copyright International Business Machines Corporation 1996, 2005. All rights reserved.

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

Contents

About this guide. xi

Who should use this guide	xii
Where to find more information	xii
Information included in your product package	xii
Support available on the Web	xiv
How to send your comments	xv
What's new in IBM DB2 Content Manager	
Enterprise Edition Version 8 Release 3	xv
What's new in DB2 Content Manager Version 8	
Release 3 for z/OS	xx

Chapter 1. Information Integrator for Content application programming concepts 1

Understanding data access through content servers	1
Understanding dynamic data object concepts	1
Dynamic data objects (DDO)	2
Extended data objects (XDO)	2
Representing multimedia content	3
Understanding content servers and DDOs	3
Comparing DDO/XDOs with attribute values and item parts	3
Understanding persistent identifiers (PID)	4

Chapter 2. Working with a federated content server and federated searching . 5

Federated schema mapping	7
Using federated content server mapping components	7
Running federated queries.	8
Federated query syntax.	9
Storing query results in federated folders (Java only)	11
Working with system administration	12
Customizing the Information Integrator for Content system administration client	12

Chapter 3. Programming with the application programming interfaces (APIs) 13

Understanding differences between the Java and C++ APIs	13
Understanding client/server architecture (Java only)	13
Packaging for the Java environment	14
Programming tips	14
Setting up the Java environment (Java only)	15
Setting the Java environment variables for Windows	15
Setting the Java environment variables for AIX	15
Setting the Java environment variables for Solaris and Linux	15
Setting the Java environment variables for z/OS USS	16
Using Remote Method Invocation (RMI) with content servers	16

Setting up the C++ environment (C++ only)	17
Setting the C++ environment variables for Windows	18
Setting the C++ environment variables for AIX	18
Building C++ programs	18
Setting the console subsystem for code page conversion on Windows	19
Understanding multiple search options	19
Tracing	20
Tracing text queries using DB2 Text Information Extender	20
Tracing parametric queries	21
Handling exceptions	21
Constants	22
Connecting to content servers	23
Establishing a connection.	23
Connecting and disconnecting from a content server in a client.	24
Setting and getting content server options	24
Listing content servers.	25
Listing the entities and attributes for a content server	26
Working with dynamic data objects (DDOs)	28
Creating a DKDDO.	28
Adding properties to a DDO	30
Creating a persistent identifier (PID)	30
Working with data items and properties.	31
Getting the DKDDO and attribute properties	33
Displaying the whole DDO	35
Deleting a DDO (C++ only)	36
Working with extended data objects (XDOs)	37
Using an XDO persistent identifier (PID)	37
Understanding XDO properties.	37
DB2 and ODBC configuration strings (C++ only)	38
Java programming tips	38
C++ programming tips	39
Programming an XDO as a part of DDO	39
Programming a stand-alone XDO	41
Examples of working with an XDO	45
Creating documents and using the DKPARTS attribute	67
Creating folders and using the DKFOLDER attribute	70
Using DKAny (C++ only)	72
Using type code	73
Managing memory in DKAny	73
Using constructors	73
Getting the type code	74
Assigning a new value to DKAny	74
Assigning a value from DKAny	74
Displaying DKAny	75
Destroying DKAny	75
Programming tips	76
Using collections and iterators	76
Using sequential collection methods	76
Using the sequential iterator.	77
Managing memory in collections (C++ only)	78

Sorting the collection	79
Understanding federated collection and iterator	80
Querying a content server	81
Differences between dkResultSetCursor and DKResults	82
Using parametric queries	82
Using text query.	88
Using the result set cursor	99
Opening and closing the result set cursor to rerun the query.	100
Setting and getting positions in a result set cursor	100
Creating a collection from a result set cursor	102
Querying collections	103
Getting the result of a query	103
Evaluating a new query	104
Using queryable collection instead of combined query	105

Chapter 4. Working with DB2 Content Manager Version 8.3 107

Understanding the DB2 Content Manager system	107
Understanding DB2 Content Manager concepts	108
Items	108
Attributes	109
Item types	109
Root and child components.	109
Objects	110
Links and references	110
Documents	111
Folders	111
Versioning	112
Access control	113
Planning a DB2 Content Manager application	116
Determining the features of your application	117
Handling errors	117
Working with the DB2 Content Manager samples	118
The insurance scenario sample	119
Creating a DB2 Content Manager application.	119
Understanding the software components	119
Representing items using DDOs	120
Connecting to the DB2 Content Manager system	120
Working with items	122
Working with folders.	148
Defining links between items	154
Working with access control	156
Creating a privilege	157
Creating a privilege set	158
Displaying privilege set properties	160
Defining an access control list (ACL)	161
Retrieving and displaying ACL information	163
Assigning an ACL to an item type	164
Assigning an ACL to an item	165
Library server and federated database limitations	166
Working with the resource manager.	167
Working with resource manager objects	167
Confidential retrieval of resource objects	168
Removing resource object contents	169
Understanding asynchronous replication in a z/OS resource manager	170
Managing documents in DB2 Content Manager	173

Creating the document management data model	174
Creating a document item type	174
Creating a document	176
Updating a document	178
Retrieving and deleting a document.	180
Versioning of parts in the document management data model	180
Working with transactions	181
Things to consider when designing transactions in your application	182
Caution when using transactions.	182
Using check-in and check-out in transactions	183
Processing transactions	183
New explicit transactions behavior in Version 8.3	184
Transaction behavior when deleting a user does not belong in a group	185

Chapter 5. Searching for data 187

Querying the DB2 Content Manager server	187
Applying the query language to the DB2 Content Manager data model	189
Understanding parametric search.	190
Understanding text search	191
Searching for object contents	192
Searching for documents	192
Making user-defined attributes text searchable	192
Understanding text search syntax.	192
Creating combined parametric and text search	194
Example searches using the query language	196
Query examples	199
Using escape sequences in your queries	206
Using escape sequences with comparison operators.	207
Using escape sequences with the LIKE operator	207
Using escape sequences with advanced text search	208
Using escape sequences with basic text search (contains-text-basic and score-basic functions)	210
Using escape sequences in Java and C++	211
Understanding row-based view filtering in query	211
Sample usage scenario	212
Description of behavior	213
Performance considerations.	214
Database Index on each filtered attribute	215
Security implications	216
The query language grammar	216

Chapter 6. Routing a document through a process 221

Understanding the document routing process	221
Understanding document routing enhancements in Version 8.3	222
Understanding Version 8.3 compatibility with Version 8.2	224
Understanding document routing classes	224
Creating document routing service objects.	227
Defining a new regular work node	228
Listing work nodes	230
Defining a new collection point	231

Defining a work list	234
Listing worklists	235
Defining a new process and associated route	236
Starting a document routing process.	239
Ending a process	240
Continuing a process.	240
Suspending a process.	241
Resuming a process	241
Listing work package persistent identifier strings in a worklist	242
Retrieving work package information	243
Listing document routing processes	244
Ad hoc routing.	245
Document routing example queries	246
Granting privileges for document routing	246
Working with access control lists for document routing	247
Programming document routing user exits	248
Document routing constants	251
Chapter 7. Understanding prefetching in DB2 Content Manager for z/OS.	253
Prefetching objects.	253
Table definitions related to prefetching	255

Chapter 8. Working with other content servers 259

Working with earlier DB2 Content Manager	261
Handling large objects	261
Using DDOs to represent earlier Content Manager content	262
Creating, updating, and deleting documents or folders.	263
Retrieving a document or folder	271
Understanding text searching (DB2 Text Information Extender)	274
Searching images by content	295
Using image search applications	298
Establishing a connection in QBIC	302
Listing image search servers	303
Listing image search databases, catalogs, and features	304
Representing image search information with a DDO	307
Working with image queries	307
Using the image search engine	311
Indexing an existing XDO using search engines	311
Using combined query	314
Understanding the earlier DB2 Content Manager workflow and workbasket functions.	318
Working with OnDemand	326
Representing OnDemand servers and documents	327
Connecting to and disconnecting from the OnDemand server.	327
Listing information on OnDemand	328
Retrieving an OnDemand document.	330
Enabling the OnDemand folder mode	337
Asynchronous search.	337
OnDemand folders as search templates.	338

OnDemand folders as native entities	338
Create and modify annotations	338
Tracing	338
Working with Content Manager ImagePlus for OS/390	340
Listing entities and attributes	340
ImagePlus for OS/390 query syntax	345
Working with DB2 Content Manager for AS/400	347
Listing entities (index classes) and attributes	347
Running a query	349
Running a parametric query	354
Working with Domino.Doc	355
Listing entities and subentities.	357
Listing cabinet attributes	359
Building queries in Domino.Doc	359
Using query syntax	359
Working with relational databases	360
Connecting to relational databases	360
Listing entities and entity attributes	362
Running a query	365
Creating custom content server connectors	368
Developing custom content server connectors	368
Using the FeServerDefBase class (Java only)	382

Chapter 9. Building Information Integrator for Content workflow applications 385

Connecting to workflow services	385
Starting a workflow	386
Terminating a workflow.	387
Listing all the workflows	388
Suspending a workflow	389
Resuming a workflow	390
Listing all the worklists	390
Accessing a worklist	391
Accessing work items	392
Moving items in the workflow	393
Listing all the workflow templates	394
Creating your own actions (Java only)	394
Working with the Information Integrator for Content workflow JavaBeans	395
Prerequisites.	395
Setting up the sample data model	396
Using the workflow JavaBeans in your application	401

Chapter 10. Building applications with non-visual and visual JavaBeans 403

Understanding basic beans concepts.	403
Using JavaBeans in builders	404
Using IBM Websphere Studio Application Developer	404
Invoking the Information Integrator for Content JavaBeans	405
Working with the non-visual beans	405
Non-visual bean configurations	406
Understanding the non-visual beans features	406
Non-visual beans categories	407
Considerations when using the non-visual beans	411
Changing locale in display names	412

Tracing and logging in the beans	413	Working with DB2 Content Manager folders through XML requests	477
Understanding properties and events for non-visual beans	413	Updating DB2 Content Manager items with an XML UpdateItemRequest	480
Building an application using non-visual beans	413	Deleting DB2 Content Manager items with DeleteItemRequest.	487
Working with visual beans	414	Checking DB2 Content Manager items out and in with CheckoutItemRequest and CheckinItemRequest	488
CMBLogonPanel bean	414	Linking DB2 Content Manager items with CreateLinkRequest or DeleteLinkRequest	490
CMBSearchTemplateList bean	416	Moving DB2 Content Manager items between entities with MoveItemRequest	492
CMBSearchTemplateViewer bean	417	Accessing DB2 Content Manager document routing using XML-based requests	493
Validating or editing fields of the CMBSearchTemplateViewer.	417	Batching multiple requests in XML requests	509
CMBSearchPanel bean	417		
CMBSearchResultsViewer bean	418	Chapter 12. Working with the Web services.	513
Overriding pop-up menus	419	Web services overview	513
CMBFolderViewer bean	419	Understanding the DB2 Content Manager Web services implementation.	515
CMBDocumentViewer bean	420	Working with the Web service in development tools	516
Viewer specifications	421	Integrating basic Web services into your applications or processes	517
Default viewers	422	Getting started with the Web services in a .NET environment.	518
Launching external viewers	422	Programming Web services requests in a .NET environment.	519
CMBItemAttributesEditor bean	422	Getting started with the Web services in a Java environment.	520
Vetoing changes in the CMBItemAttributesEditor	423	Programming Web services requests in a Java environment.	521
CMBVersionsViewer bean	423	Authenticating Web services requests for security	523
General behaviors for visual beans	423	Creating a new instance of an item through Web services	524
Replacing a visual bean	424		
Building an application using visual beans	425	Chapter 13. Working with the Java document viewer toolkit.	529
Chapter 11. Working with XML services (Java only)	429	Viewer architecture	530
Understanding how XML services work with other DB2 Content Manager programming layers	429	The document engines	531
Importing and exporting DB2 Content Manager metadata using XML services	432	The annotations engine	532
Importing and exporting administration objects as XML	433	Example viewer architectures	532
Importing and exporting DB2 Content Manager data model objects as XML schema files (XSD)	434	Creating a document viewer	534
Importing and exporting DB2 Content Manager data instance objects as XML	454	Creating a standalone viewer application or applet	534
Exporting DB2 Content Manager DDO items as XML items	455	Working with documents and annotations.	536
Importing XML items as DB2 Content Manager DDO items	456	Customizing the viewer	537
Importing and exporting XML object dependencies	457	Working with the annotation services	543
Extracting content from different XML sources	458	Using annotation services interfaces	543
Mapping a user-defined schema to a storage schema with the XML schema mapping tool	458	Understanding annotation editing support	544
Programming runtime operations through the XML JavaBeans	461	Building an application using the annotation services	545
Listing DB2 Content Manager servers with ListServerRequest	464	Adding a custom annotation type to your application	545
Authenticating Web service requests for security	465	Working with the page manipulation functions	546
Changing a password with XML requests	466		
Listing DB2 Content Manager entities with ListSchemaRequest	466		
Creating DB2 Content Manager items with CreateItemRequest	468		
Searching DB2 Content Manager items with RunQueryRequest	469		
Retrieving DB2 Content Manager items with RetrieveItemRequest	472		
Viewing your user privileges with XML requests	476		

Chapter 14. Working with the JSP tag library and controller servlet 549

Setting up the tag library and servlet	549
Using the tag library	549
Conventions used in the tag library	550
Tag summary	550
Connection related tags	550
Schema related tags	551
Search related tags	552
Item related tags	552
Folder related tags.	553
Document related tags	553
Information Integrator for Content controller servlet.	554
What the servlet can do	554
Servlet reference	555
Servlet toolkit function matrix	560

Chapter 15. Troubleshooting 563

Receiving an error when compiling C++ applications that are Unicode enabled	563
Receiving an error when using reference attributes	563
Cannot add, store, retrieve, or update a resource item	564
Cannot import a DKDDO object from XML	564
Receiving an error when updating, reorganizing, or using text indexes for text searchable components	565

Notices 567

Trademarks	569
----------------------	-----

Glossary 571

Index 581

About this guide

This guide describes how to use the Java™, JavaBeans™, and C++ application programming interfaces (APIs) provided with Information Integrator for Content Version 8 Release 3 and DB2 Content Manager Version 8 Release 3. The APIs and beans provide building blocks for creating applications that access content stored in heterogeneous content servers.

In earlier versions, DB2 Content Manager and Information Integrator for Content kept separate application programming guides. In Version 8 Release 3, the two products share many of the APIs and are based on many of the same programming concepts. Also, because much of the new functionality in Information Integrator for Content Version 8 Release 3 involves the new connector to DB2 Content Manager Version 8 Release 3, the two guides merged into one.

This guide includes:

- An introduction to Information Integrator for Content and DB2 Content Manager application programming concepts, including dynamic data object concepts in the context of Java and C++
- A description of the function accessible through the DB2 Content Manager Version 8 Release 3 connector
- Documentation on all other Information Integrator for Content 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 DB2 Content Manager imply both pre-Version 8.1 and Version 8 Release 3 of the product.

Important information for Linux users:

This manual is provided for your reference and might contain documentation about the following components not supported on Linux:

- IBM DB2 Information Integrator for Content
- DB2 Information Integrator for Content advanced workflow
- All C++ connectors
- Remote Java connectors
- IBM Content Manager Version 7 Release 1 connector
- Content Manager ImagePlus for OS/390® connector
- Lotus® Domino.Doc® connector
- Relational database connectors (DB2® UDB, JDBC, ODBC)

The specific product versions in the list below are supported on the Linux platform:

- Content Manager OnDemand for Multiplatforms Version 7 Release 1
- Content Manager OnDemand for iSeries Version 5 Release 1 and Version 5 Release 2
- Content Manager OnDemand for z/OS® and OS/390® Version 2 Release 1 and Version 7 Release 1

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 provides access to 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 a browser-based information center and publications in portable document format (.PDF).

The information center

The product package contains an Eclipse-based information center that you can install when you install the product. You can install and use the information center locally on a single workstation or you can install it on a server. The information center is also available on the Internet at:

<http://publib.boulder.ibm.com/infocenter/cm83>

Starting the information center: The information center automatically starts as a service/daemon Web server on the machine where you install it. You can view it from a different machine, by launching a Web browser and entering: <http://hostname:8081>. You can view it from the same machine, by launching a Web browser and entering: <http://localhost:8081>. (The default port number is 8081, but you can change it in the IBMCMROOT/config/cmcfgrc.ini file.)

If the information center fails to start as a service/daemon, you can manually start it by running the IBMCMROOT/infoctr/web-start script. The process will continue to run in the background until you run the IBMCMROOT/infoctr/web-end script. You can also start the information center with a randomly generated port number (known as standalone client mode) by running the IBMCMROOT/infoctr/local-start script (local-end stops the information center). You cannot run these two modes simultaneously.

Information center content: Depending on what you install, the information center includes some or all of the documentation for DB2 Content Manager, DB2 Information Integrator for Content, and DB2 Content Manager VideoCharger™.

When you open the information center, the Welcome page displays. The Welcome page provides links to product orientation material both within the information center and beyond, and also provides instructions for using the information center effectively. The Welcome page links to information roadmaps, which in turn link to task-specific information both within the information center and beyond.

Within the information center, information is organized by task (for example, Administering). In addition to the provided navigation mechanism and indexes, a search facility also aids retrievability. When you use a local version, you can also take advantage of bookmarks to return to frequently used topics.

When you first open the local or intranet information center, it does not include the planning and installation information. Instructions are provided for downloading this material and including it in your information center. You can review this material at any time by viewing the information center on the Web at: <http://publib.boulder.ibm.com/infocenter/cm83>

In Version 8.3, the information center includes the DB2 Content Manager and DB2 Information Integrator for Content system administration help. When system administrators click **Help** in the System Administration Client, the appropriate topic displays in of the information center in context with all other system administration information. The content of the DB2 Content Manager PDF publication, *System Administration Guide*, is identical to the content displayed in the information center.

Also in Version 8.3, the Javadoc application programming reference information, which was formerly displayed in EDO (Enterprise Documentation Online) is included in the information center. The samples (Java, C++, and Web services) are also included in the information center.

PDF publications

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

Table 1 shows the publications for IBM DB2 Content Manager Enterprise Edition (which includes DB2 Information Integrator for Content and DB2 Content Manager VideoCharger) and IBM DB2 Content Manager for z/OS.

Table 1. IBM DB2 Content Manager Version 8.3 solution publications

Path\file name	Title	For products:	Publication number
CM\admincm	<i>System Administration Guide</i>	<ul style="list-style-type: none"> • DB2 Content Manager Enterprise Edition • DB2 Content Manager for z/OS • DB2 Information Integrator for Content 	SC27-1335-06
CM\apg	<i>Application Programming Guide</i>	<ul style="list-style-type: none"> • DB2 Content Manager Enterprise Edition • DB2 Content Manager for z/OS • DB2 Information Integrator for Content 	SC27-1347-04
CM\vecliinst	<i>Installing, Configuring, and Managing the eClient</i>	<ul style="list-style-type: none"> • DB2 Content Manager Enterprise Edition • DB2 Content Manager for z/OS • DB2 Information Integrator for Content 	SC27-1350-05

Table 1. IBM DB2 Content Manager Version 8.3 solution publications (continued)

Path\file name	Title	For products:	Publication number
CM\messcode	<i>Messages and Codes</i>	<ul style="list-style-type: none"> DB2 Content Manager Enterprise Edition DB2 Content Manager for z/OS DB2 Information Integrator for Content 	SC27-1349-05
Not on the documentation CD ¹	<i>Migrating to DB2 Content Manager Version 8</i>	DB2 Content Manager Enterprise Edition	SC27-1343-02
Not on the documentation CD ¹	<i>Migrating to DB2 Content Manager Version 8 for z/OS</i>	DB2 Content Manager for z/OS	GC18-7699-01
CM\wincli	<i>Client for Windows Programming Reference</i>	<ul style="list-style-type: none"> DB2 Content Manager Enterprise Edition DB2 Content Manager for z/OS DB2 Information Integrator for Content 	SC27-1337-03
Not on the documentation CD ¹	<i>Planning and Installing Your Content Management System</i>	<ul style="list-style-type: none"> DB2 Content Manager Enterprise Edition DB2 Information Integrator for Content 	GC27-1332-03
Not on the documentation CD ¹	<i>Planning and Installing Your Content Management System for z/OS</i>	DB2 Content Manager for z/OS	GC18-7698-01
VC\adminvc	<i>System Administration Guide</i>	DB2 Content Manager VideoCharger	SC27-1351-03
VC\installvc	<i>Planning and Installing DB2 Content Manager VideoCharger</i>	DB2 Content Manager VideoCharger	GC27-1353-03
VC\vcprogref	<i>Programmer's Reference</i>	DB2 Content Manager VideoCharger	SC27-1352-03

Note:

1. Available for viewing here: <http://publib.boulder.ibm.com/infocenter/cm83>
Also available for download.

Support available on the Web

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

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

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

www.ibm.com/software/data/videocharger/

The documentation is included in softcopy with the product. To access product documentation on the Web, go to the IBM Publications Center at www.ibm.com/shop/publications/order. The IBM Publications Center is also linked from the product support pages.

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 DB2 Content Manager, DB2 Information Integrator for Content, or DB2 Content Manager VideoCharger documentation. You can use either of the following methods to provide comments:

- Send your comments from the Web. Visit the IBM Data Management Online Reader's Comment Form (RCF) page at:
www.ibm.com/software/data/rcf
You can use the page to enter and send comments.
- Send your comments by e-mail to comments@us.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).

What's new in IBM DB2 Content Manager Enterprise Edition Version 8 Release 3

Version 8.3: Version 8.3 continues to deliver a real return on investment to customers. Version 8.3 focuses on five areas: integration, open systems, autonomic systems, resiliency, and ease of use. These highlights, and other enhancements to the Version 8.3 product, are summarized below:

Support for Oracle databases

DB2 Content Manager Version 8.3 adds support for Oracle databases managing the metadata stored in both library server and resource manager. Migration tools are included for Oracle users of DB2 Content Manager Version 7.

Remote database server for DB2 Universal Database™ and Oracle

You can now help reduce workload by installing the DB2 Content Manager resource manager database on a different machine than the resource manager application.

Web services support

DB2 Content Manager provides a self-contained, self-describing modular interface, a Web services interface, that you can use within your applications, with other Web services, or in complex business processes to seamlessly access items stored in DB2 Content Manager. The Web services interface allows you to integrate dynamically your applications with DB2 Content Manager, regardless of the programming language they were written in and the platform they reside in.

XML support

The system administration client enables you to export your system administration data into an XML readable file and import that data from an XML file. This capability allows you to copy administrative settings from one server to another by exporting the information and importing it into the systems. You can also use the new XML capability to get the list of system administration objects from one DB2 Content Manager or DB2 Information Integrator for Content system to another.

Document routing enhancements

DB2 Content Manager document routing is enhanced in Version 8.3 to include decision points, actions, action lists, parallel routing, and user

exit support. In addition, a new graphical builder within the system administration client helps you easily define your document routing processes.

Query (search) enhancements

The query function is enhanced to include the following support:

- Query on checked-out items
- Row-based view filtering in query
- Get the count of query results without getting the results themselves
- Use of the IN operator to compare an attribute's value to a list of values
- Internal query optimization to reduce the length of the generated SQL statements

Similar characteristics for logging and tracing

Version 8.3 provides logs with similar characteristics, which cover most system components:

- The system administration client now provides the log control utility, which you can use to set log and trace parameters for multiple system components
- A default common directory for all log files
- A standard log file timestamp format using Greenwich Mean Time
- Logging information related to a single user ID
- A unique log ID that is common among across different system component log files

Installation improvements

- The installation programs for DB2 Content Manager, DB2 Information Integrator for Content, and DB2 Content Manager eClient are redesigned to provide commonality for all operating systems, consistent product interoperability, and an improved, more robust, installation experience.
- You can selectively install features of the products and some features are sharable among the products.
- The installation programs for DB2 Content Manager and DB2 Information Integrator for Content now include time saving Typical installation paths, which greatly reduce the complexity of user input for common installations. The Custom paths are reorganized to improve clarity and consistency.
- Silent installation capability is consistently supported for all products and operating systems, allowing for the full range of installation options.
- Pre-requisite checking is redesigned to be more flexible and precise, thus facilitating a wider range of installation topologies, and allowing the flexibility to extend capabilities.
- Version 8.3 now automatically configures the Secure Sockets Layer (SSL) of the IBM HTTP Server shipped with WebSphere® Application Server Version 5.1 and used by the DB2 Content Manager resource manager.
- Online help is available for installation panels that provides information about default values and limitations for fields, and relevant background information.

Discontinued and deprecated function

The following function is no longer supported in DB2 Information Integrator for Content in V8.3:

- Extended Search
- IBM Content Connector for Panagon Image Services
- Information mining
- IBM Web Crawler
- Connectors for:
 - Information Catalog Manager
 - Extended Search
 - DataJoiner[®]
 - ActiveX versions of the following connectors: Content Manager V7 (DL), Lotus Domino Doc (DD), OnDemand (OD), Extended Search (DES), VisualInfo[™]/400[®] (V4), Image Plus/390(IP) and Federated (Fed))

Accessibility improvements

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. Version 8.3 enhances product accessibility features. For example, new shortcuts have been added to help you operate all features using the keyboard instead of the mouse.

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

Support for Linux

DB2 Content Manager V8.2 now supports Linux. The following changes are also new for the Linux-supported product:

- The system administration client runs natively on Linux.
- You are not required to have a C compiler on your library server machine.
- You can install the library server and resource manager on Linux.

Information Integrator for Content name change to IBM Information Integrator for Content

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

Replication

DB2 Content Manager V8.2 includes resource manager replication, which is the ability to store objects in multiple locations, managed by replication resource managers. Object replicas will behave as LAN cache objects for improved load balancing.

LAN cache

LAN cache support in DB2 Content Manager V8.2 provides application-transparent caching, using local servers as defined by the system administrator.

Support for DB2 UDB V8.1

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

Support for WebSphere Application Server Version 4 and Version 5

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

Federated folders

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

Advanced workflow collection points

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

Microsoft® Visual Studio .NET for building applications

The DB2 Content Manager and Information Integrator for Content 8.1 and later APIs now support Microsoft Visual Studio .NET for writing content management applications or to integrate applications built using Microsoft Visual Studio .NET.

Version 8.1: Version 8.1 begins a legacy of integration and versatility. One of the many highlights and improvements from previous Content Manager products is the new data model structure which allows for more document customization. The changes to the DB2 Content Manager product in Version 8.1 are summarized below:

Improved performance

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

Support for Sun Solaris

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

Enhanced data model

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

Improved workflow

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

Integrated text search

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

Common system administration

A single client application provides separate access to Content Manager and Information Integrator for Content. 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.

Information center

The browser-based information center includes the documentation for DB2 Content Manager, Information Integrator for Content, and Content Manager VideoCharger. Topic-based information is organized by product and by task (for example, Administration). In addition to the provided navigation mechanism and indexes, a search facility also aids retrievability.

Accessibility

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

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

PeopleSoft and Siebel integrations

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

What's new in DB2 Content Manager Version 8 Release 3 for z/OS

Version 8.3 for z/OS: Version 8.3 continues to deliver a real return on investment to customers. Version 8.3 focuses on five areas: integration, open systems, autonomic systems, resiliency, and ease of use. These highlights, and other enhancements to the Version 8.3 product, are summarized below:

DB2 Content Manager for z/OS support for Tivoli® Storage Manager application programming interfaces

In Version 8.3 the capabilities of the DB2 Content Manager resource manager for z/OS expands its storage options with the introduction of Tivoli Storage Manager support. Tivoli Storage Manager support allows customers to not only store objects to a DB2 datastore using the Object Access Method (OAM), but also leverage the function provided by the Tivoli Storage Manager application programming interfaces. Storing objects to a resource manager on z/OS and a Tivoli Storage Manager collection increases the maximum object size currently supported and offers a higher degree of concurrency than can be achieved using OAM.

IBM DB2 Content Manager Toolkit for z/OS

The DB2 Content Manager Toolkit provides connectivity to Content Manager systems running on all supported platforms to clients running under z/OS UNIX®. These clients include Java applications that run in batch under z/OS UNIX or Web applications that run in the WebSphere for z/OS application server.

Query (search) enhancements

The query function now includes the following support:

- Query on checked-out items
- Row-based view filtering in query
- Get the count of query results without getting the results themselves
- Use of the IN operator to compare an attribute's value to a list of values
- Internal query optimization to reduce the length of the generated SQL statements

Document routing enhancements

DB2 Content Manager document routing is enhanced in Version 8.3 to include decision points, actions, action lists, parallel routing, and user exit support. In addition, a new graphical builder within the system administration client helps you easily define your document routing processes.

Similar characteristics for logging and tracing

Version 8.3 provides logs with similar characteristics, which cover most system components:

- The system administration client now provides the log control utility, which you can use to set log and trace parameters for multiple system components.
- A default common directory for all log files.
- A standard log file timestamp format using Greenwich Mean Time.
- Logging information related to a single user ID.
- A unique log ID that is common among across different system component log files.

Version 8.2 for z/OS and OS/390: Content Manager Version 8.2 and higher extends a legacy of integration and versatility to z/OS and OS/390 customers. One of the many highlights and improvements from Content Manager Version 2.3 and ImagePlus Version 3.1 is the new data model structure, which allows for more document customization. The changes to the DB2 Content Manager product in Version 8.2 and later for z/OS and OS/390 are summarized below:

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.

Continued use of OAM

ImagePlus for OS/390 stores objects using the Object Access Method (OAM), which is now a component of DFSMS. Content Manager Version 8.2 and later for z/OS and OS/390 will continue to use OAM for object storage and retrieval, so movement of actual objects for migration is unnecessary. In addition, you can continue your current strategy for using OAM, including the OSMC component.

Improved performance over Content Manager Version 2.3 for OS/390

The library server and resource manager use DB2 stored procedures and leverage DB2 technology to significantly reduce network traffic and improve performance and scalability. The resource manager for OS/390 is implemented as a fast CGI program running on the IBM HTTP Server; WebSphere Application Server for OS/390 is not required.

Common system administration

A single client application provides separate access to Content Manager and Information Integrator for Content. 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. The Client for Windows supplies direct access to Content Manager Version 8.2 and later for z/OS and OS/390. This client supports Content Manager Version 8 document routing, the hierarchical data model (to a single child component level), versioning, and it provides federated access to Content Manager Version 8 and ImagePlus for OS/390 servers.

The Web-based eClient also supplies direct access to Content Manager for z/OS and OS/390 servers and includes Information Integrator for Content advanced workflow, version control, multi-valued attributes, and federated access to heterogeneous backend server types, including ImagePlus for OS/390.

Migration utilities

Utilities to aid migration are available for customers who have either Content Manager Version 2.3 for OS/390 with applied PTFs or the following ImagePlus product components:

- Object Distribution Manager Version 2.2 or Version 3
- Folder Application Facility/Application Programming Interface Version 2.2 or Version 3

- Folder Application Facility/Folder Workflow Application Version 2.2.1 or Version 3

You do not migrate data from Content Manager Version 8.2 to DB2 Content Manager Version 8.3.

Uses UNIX System Services

CICS[®] is not required for Content Manager Version 8.2 and later.

Chapter 1. Information Integrator for Content application programming concepts

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

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.

Information Integrator for Content supports the following content servers:

- DB2 Content Manager Version 8 Release 3
- DB2 Content Manager Version 7 Release 1
- Domino.Doc
- ImagePlus for OS/390
- Content Manager OnDemand
- VisualInfo for AS/400®
- DB2 UDB
- JDBC/ODBC servers

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

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

Schema mapping functions provide the schema information for each content server. The information provided by schema mapping is used during federated searching, federated collection, and DB2 Information Integrator for Content system administration. Information Integrator for Content 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, Information Integrator for Content provides an implementation of the dynamic data object (DDO) and its extension, the extended data object (XDO), which are part of the CORBA Persistent Data Service (PDS) protocols. The concepts of DDO and XDO are not specific to any one

content server, and can be used to represent data objects in any database management system supported by Information Integrator for Content.

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

Dynamic data objects (DDO)

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

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

For example, a DDO can represent a row of a database table whose columns are represented by DDO's data items and their properties. A DDO can contain one or more extended data objects (XDOs) that represent non-traditional data types. Figure 1 shows dynamic data objects and data items.

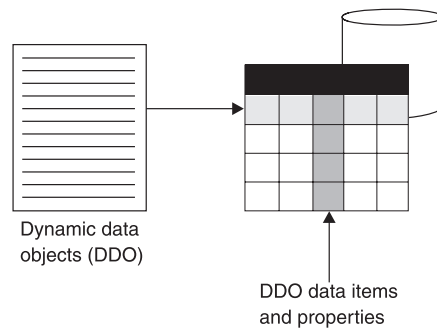


Figure 1. Dynamic data objects and items

Extended data objects (XDO)

An XDO is a representation of complex multimedia data, for example a resource item storing an image or document in DB2 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 2 on page 3 shows an example of XDOs.

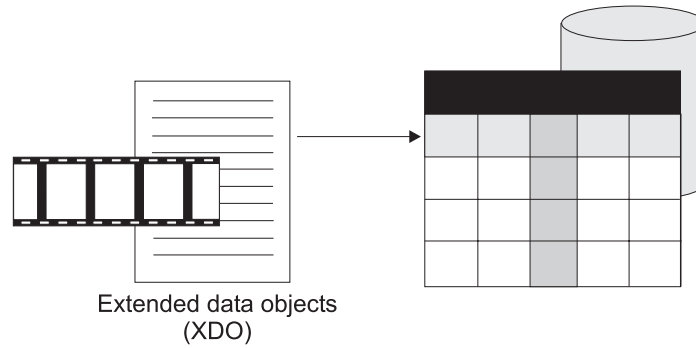


Figure 2. Extended data objects (XDOs)

Representing multimedia content

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

In DB2 Content Manager 8.3, a DDO consists of all the metadata that describes the object, such as a document or image. An XDO extends the DDO functionality further to support resource content. Resource content is any type of content ranging from binary data or text to video and audio streams. An item that implements (and should be) an XDO is also a DDO and supports all of the functionality provided by a DDO plus the additional functionality that is provided by (and should be) an XDO.

Understanding content servers and DDOs

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

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

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

Comparing DDO/XDOs with attribute values and item parts

A DDO corresponds to an item in Information Integrator for Content. The DDO's object type corresponds to the item's associated item type. The data items of a DDO correspond to an item's attributes. For example, in DB2 Content Manager an item type is created using a set of attributes, and an item is always indexed by an item type.

A DDO can hold one or more XDOs that correspond to item parts in Information Integrator for Content.

Understanding persistent identifiers (PID)

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

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

Chapter 2. Working with a federated content server and federated searching

Federated searching is the process of searching for data in one or more content servers. You use a DKDatastoreFed object for a federated search. Federated search works with classes that are specific implementations of dkDatastore, dkDatastoreDef, and other related classes that support federated searches. The specific federated classes work together with other common classes, such as those for queries, collections, and data objects and are part of the Information Integrator for Content 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 3.

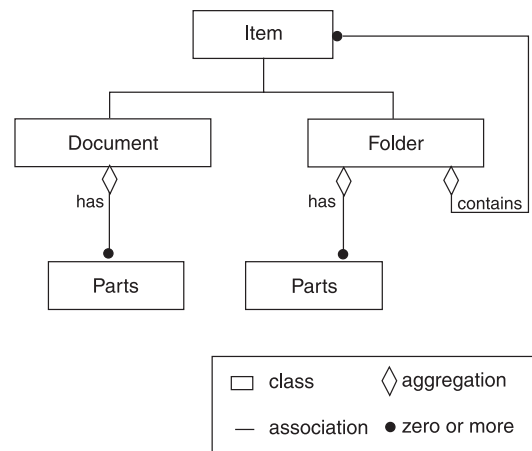


Figure 3. Federated document view

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

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

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

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

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

Table 2. Mapping terminology for each content server

Content server	Data source	Entity	Attribute	View
DB2 Content Manager 8.3	library server	item type	attribute	item type view or item type subset
DB2 Content Manager 7.1	library server	index class	<ul style="list-style-type: none"> • attribute • key attribute 	index class view
OnDemand	OnDemand server	<ul style="list-style-type: none"> • application group • folder 	<ul style="list-style-type: none"> • field • criteria 	N/A
ImagePlus	ImagePlus for OS/390 server	entity	attribute	N/A
DB2 Content Manager for AS/400	DB2 Content Manager for AS/400 server	index class	attribute	index class view
Domino.Doc	Domino server	library room cabinet binder	profile field keyword	N/A
Relational database	IBM DB2 UDB, JDBC, ODBC,	table	column	view
Federated content server	mapping server	mapped federated entity	mapped federated attribute	search template
Federated content server that can hold federated folders	server	federated entity	federated attribute	federated folder

Figure 4 on page 7 illustrates a federated search. The federated search uses the Information Integrator for Content federated content server, working through search templates. The federated content server then calls the searches for the individual content servers to perform the actual search on the content servers. This association is established by schema mapping.

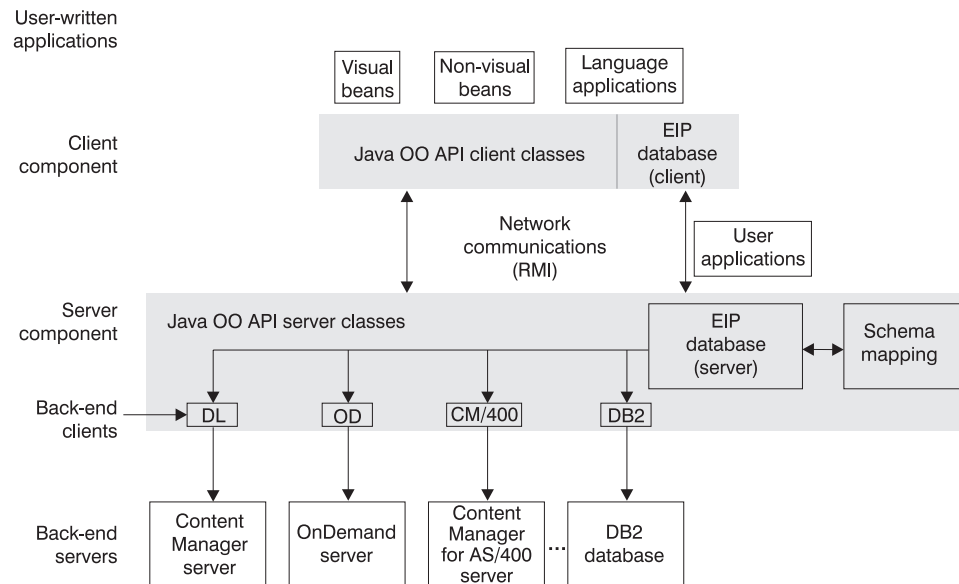


Figure 4. Structure of federated searches

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

Federated schema mapping

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

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

Using federated content server mapping components

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

User ID and password mapping

To support a single logon feature, each user ID in the Information Integrator for Content 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 Information Integrator for Content.

The user ID and content server information is maintained in the Information Integrator for Content administration database.

Running federated queries

To run a federated search using the APIs, start by creating a federated query string. You can then create and run the query by passing the query string to the `execute` or `evaluate` method of the federated content server to process the query directly.

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

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, Information Integrator for Content performs the following steps:

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

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

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

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

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

The components of a federated search and their relationships are illustrated in Figure 5 on page 9.

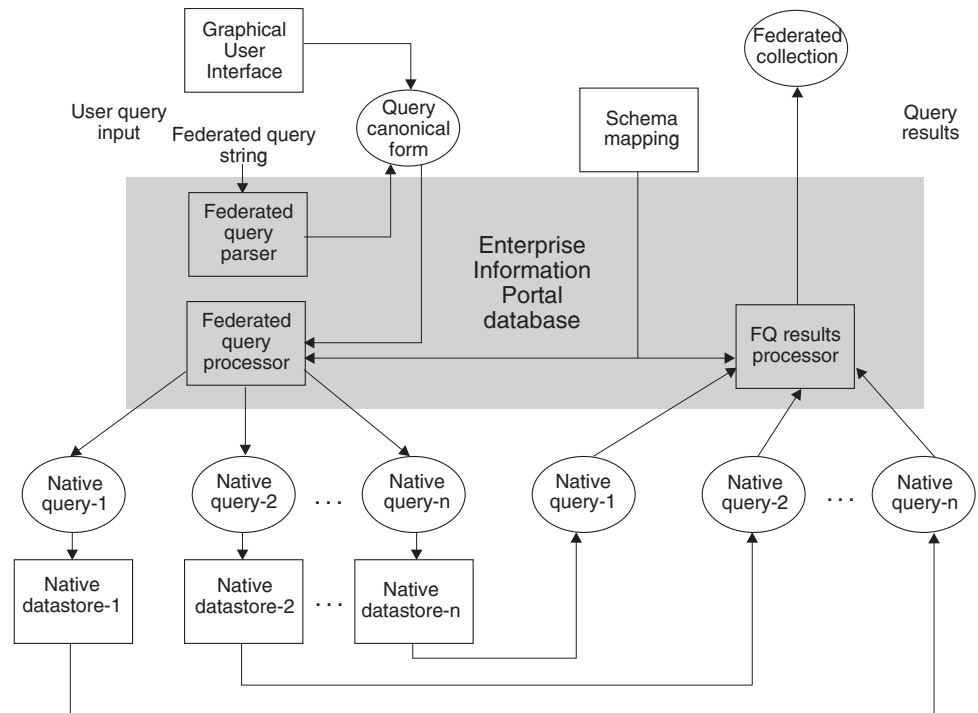


Figure 5. Federated query processing

Federated query syntax

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

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

[and

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

]

The NOT operator is not supported in federated searches.

Examples of federated query strings

Federated parametric query using the LIKE operator

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

Federated parametric query using the LIKE and > operator

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

```

Federated parametric query using the LIKE and < operator

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

```

Federated parametric query using the BETWEEN operator

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

```

MAX_RESULTS returns all results when set to zero.

Federated parametric query using the NOTBETWEEN operator

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

```

Federated parametric query using the IN operator

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

```

Federated parametric query using the NOTIN operator

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

```

Federated parametric query using the == operator

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

```

Federated parametric query using the <> operator

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

```

Federated parametric query using the AND and OR operators

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

```

Federated parametric query using the CONTAINS_TEXT_IN_CONTENT operator

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

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

```

Federated parametric query using the CONTAINS_TEXT operator

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

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

```

Federated text query

Searches for text in content. The text can be a word or a phrase. The ASSOCIATED_ENTITY keyword is only applicable when a federated entity is text-searchable. This is only valid when the text-searchable federated

entity (FedEntity) is mapped to a Content Manager Version 8 text-searchable item type or an Extended Search text-searchable entity.

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

Federated text query

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

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

Federated text query

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

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

Federated parametric and text query using the OR operator

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

Federated parametric and text query using the AND operator

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

Federated parametric and text query on attributes using the OR operator

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

Federated parametric and text query on attributes using the OR operator

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

Storing query results in federated folders (Java only)

Enterprise Information Portal Version 8 Release 3 now provides special federated entities that can hold *federated folders*. These federated folders can store the combined results from a federated query, such as a document from Content Manager and a related document from OnDemand. You can then send the results directly into a workflow.

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

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

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

For code samples, see the `samplesdirectory`.

Working with system administration

Enterprise Information Portal provides the classes and APIs for you to access system administration functions. See the Application Programming Reference for information on the specific classes.

Customizing the Information Integrator for Content system administration client

The Information Integrator for Content system administration client supports extending the system administration application to include custom functions:

- You can replace the user and user group dialogs in the Information Integrator for Content system administration client with your own dialogs.
- You can add new nodes to the hierarchy in the Information Integrator for Content 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 Information Integrator for Content system administration.

Chapter 3. Programming with the application programming interfaces (APIs)

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

The APIs support:

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

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

Understanding differences between the Java and C++ APIs

The list below describes differences between the IBM DB2 Information Integrator for Content Java and C++ API sets:

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

Understanding client/server architecture (Java only)

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

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

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

- The client package is `com.ibm.mm.sdk.client`. The classes in the client package communicate with the classes in the server package via RMI.
- The common classes are shared by both the client and server. Sometimes an application does not know where the content resides. For example, an application can have content residing on the client at one time and the server at another time. The `cs` package connects the client and server dynamically.

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

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

Packaging for the Java environment

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

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

Access and manipulate content server information

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

Communicate with the server package using Remote Method Invocation (RMI)

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

Common classes for both the server package, client package, and the `cs` package

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

Connect the client or server dynamically

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

Programming tips

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

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

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

Setting up the Java environment (Java only)

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

server package

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

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

client package

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

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

cs package

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

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

Setting the Java environment variables for Windows

You can run cmbenv81.bat in a Windows command prompt to set up the environment.

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

PATH Ensure your PATH contains %IBMCMROOT%

CLASSPATH

Make sure your CLASSPATH contains %IBMCMROOT%

Setting the Java environment variables for AIX

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

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

PATH Make sure your PATH contains /opt/IBM/db2cmv8

LIBPATH

Make sure your LIBPATH contains /opt/IBM/db2cmv8

LD_LIBRARY_PATH

Make sure your LD_LIBRARY_PATH contains /opt/IBM/db2cmv8/lib

CLASSPATH

Make sure your CLASSPATH contains /opt/IBM/db2cmv8/lib/xxx where xxx are the .jar files, (for example, cmbfed81.jar)

Setting the Java environment variables for Solaris and Linux

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

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

PATH Make sure your PATH contains /opt/IBM/db2cmv8/lib

LIBPATH

Make sure your LIBPATH contains /opt/IBM/db2cmv8/lib

LD_LIBRARY_PATH

Make sure your LD_LIBRARY_PATH contains /opt/IBM/db2cmv8/lib

CLASSPATH

Make sure your CLASSPATH contains /opt/IBM/db2cmv8/lib/xxx where xxx are the .jar files, (for example, cmbfed81.jar)

Setting the Java environment variables for z/OS USS

In the z/OS USS environment, you can use a shell script (cmbenv81.sh) to set up your development environment for developing DB2 Information Integrator for Content applications.

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

PATH

Ensure that your PATH contains \$IBMCMROOT/cmgmt, \$IBMCMROOT/cmgmt/connectors, \$IBMCMROOT/lib

LIBPATH

Ensure that your LIBPATH contains \$IBMCMROOT/lib

LD_LIBRARY_PATH

Ensure that your LD_LIBRARY_PATH contains \$IBMCMROOT/lib CLASSPATH

Ensure that your CLASSPATH contains \$IBMCMROOT/lib/xxx where xxx are the .jar files, for example, cmbcm81.jar.

Using Remote Method Invocation (RMI) with content servers

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

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

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

To start the RMI server, use cmbregist81.bat on Windows or cmbregist81.sh on AIX or Solaris. Before starting the RMI server, define the correct port number and server type. For information on configuring and administering RMI servers, see *Planning and Installing Information Integrator for Content and Managing Information Integrator for Content*.

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

When you set up your Windows or AIX environment, you must establish the settings described in this section. Table 3 lists Library, AIX shared and DLL requirements.

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

Attention: cmbcm81x.lib is for release build and cmbcm81xd.lib is for debug build, where *x* represents either Microsoft Visual C++ .net Version2002 (7) or Microsoft Visual C++ .net Version2003 (71) compiler.

Also, to compile DB2 Content Manager applications using the .NET 2003 compiler, you must add the MSVC71 compile flag in the .mak files or in the project files. For example, in the compile flag definitions, add /D MSVC71. If you do not define the compiler flag, your application will not compile.

Table 3. Shared objects and DLL environment information

Connector	Library	Windows DLLs	Shared objects for AIX
Common Note: This is not a connector. It is a common set of APIs used by all of the connectors.	cmbcm81x.lib cmbcm81xd.lib	cmbcm81x.dll cmbcm81xd.dll	libcmbcm816.a
DB2 Content Manager Version 8.3	cmbicm81x.lib cmbicm81xd.lib	cmbicm81x.dll cmbicm81xd.dll cmbicmfac81x.dll cmbicmfac81xd.dll	libcmbicm816.a libcmbicmfac816.so
Content Manager Version 7.1	cmbdl81x.lib cmbdl81xd.lib	cmbdl81x.dll cmbdl81xd.dll cmbdlfac81x.dll cmbdlfac81xd.dll de_db2.dll de_db2_d.dll de_ora.dll de_ora_d.dll	libcmbdl816.a libcmbdlfac816.so
Federated	cmbfed81x.lib cmbfed81xd.lib	cmbfed81x.dll cmbfed81xd.dll cmbfedfac81x.dll cmbfedfac81xd.dll	libcmbfed816.a libcmbfedfac816.so
DB2 Universal Database Version 8.1	cmbdb281x.lib cmbdb281xd.lib	cmbdb281x.dll cmbdb281xd.dll cmbdb2fac81x.dll cmbdb2fac81xd.dll	libcmbdb2816.a libcmbdb2fac816.so
ODBC	cmbodbc81x.lib cmbodbc81xd.lib	cmbodbc81x.dll cmbodbc81xd.dll cmbodbcfac81x.dll cmbodbcfac81xd.dll	Not supported

Table 3. Shared objects and DLL environment information (continued)

Connector	Library	Windows DLLs	Shared objects for AIX
OnDemand	cmbod81x.lib cmbod81xd.lib	cmbod81x.dll cmbod816xd.dll cmbodfac81x.dll cmbodfac81xd.dll	libcmbod816.a libcmbodfac816.so
ImagePlus for OS/390	cmbip81x.lib cmbip81xd.lib	cmbip81x.dll cmbip81xd.dll cmbipfac81x.dll cmbipfac81xd.dll	Not supported
VisualInfo	cmbv481x.lib cmbv481xd.lib	cmbv481x.dll cmbv481xd.dll cmbv4fac81x.dll cmbv4fac81xd.dll	Not supported
Domino.Doc [®]	cmbdd81x.lib cmbdd81xd.lib	cmbdd81x.dll cmbdd81xd.dll cmbddf81x.dll cmbddf81xd.dll	Not supported

Setting the C++ environment variables for Windows

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=%PATH%;x:\%IBMCMROOT%\DLL if you want to include the DLL directory to PATH.

INCLUDE

set INCLUDE=%INCLUDE%;x:\%IBMCMROOT%

Setting the C++ environment variables for AIX

See the sample MAK files in the samples directory for more information.

Set the following environment variables:

In the AIX environment, you can use the cmbenv81.sh batch file to set up your development environment.

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

NLS path

export NLSPATH=\${NLSPATH}:/opt/IBM/db2cmv8/msg/En_US/%N

PATH

PATH=\${PATH}:/opt/IBM/db2cmv8/lib

LIBPATH

export LIBPATH=\${LIBPATH}:/opt/IBM/db2cmv8/lib

Building C++ programs

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

When building an application using the C++ APIs that you will use for debugging, link your application with the debug version of the API libraries, that is, the

d.lib** or ***.lib** if you are on Windows. If you are on AIX, link to **lib816.a. There is no debug build on AIX since it is not needed.

Restriction: The Content Manager V8 C++ APIs do not support Unicode. For more information see “Receiving an error when compiling C++ applications that are Unicode enabled” on page 563.

Working with Microsoft Visual Studio .NET

The DB2 Information Integrator for Content and DB2 Content Manager versions 8.2 and later APIs support Microsoft Visual Studio .NET. When using Microsoft Visual Studio .NET to build your applications, however, you must link to the appropriate library, which is determined by the connector and Visual Studio C++ version you are using, at compile time.

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

Table 4. Microsoft Visual Studio .NET library names

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

Setting the console subsystem for code page conversion on Windows

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

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

Understanding multiple search options

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

Content Manager Version 8 offers an integrated text feature, which no longer requires a separate text search facility (earlier Content Manager still does). See “Understanding text search” on page 191.

Use the multiple search options to search within a given content server, using one or a combination of supported queries, listed below, or search on the results of a

previous search. Each search type is supported by one or more search engines. Not all content servers support multiple search options.

Parametric search

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

Text search

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

Note: The DB2 Net Search Extender was named DB2 Text Information Extender in earlier versions of DB2 Universal Database.

Image search

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

Combined search

Searches using both parametric and text search.

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

Tracing

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

Tracing text queries using DB2 Text Information Extender

The DB2 Text Information Extender (TSE) and all of its functions can only be used with earlier DB2 Content Manager. Content Manager Version 8 offers an integrated text feature, which does not require a separate text search facility. See "Understanding text search" on page 191.

The following environment variable setting writes the trace for a DB2 Text Information Extender 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 DB2 Text Information Extender 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:

`CMBMTMTERM=fileName` (for example, `.\tmterm.out`)

Note: DB2 Content Manager Version 8 uses an integrated text search. If you are using earlier IBM Content Manager, you can still use the Text Search Engine (TSE).

Tracing 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`

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

Handling exceptions

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

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

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

```
Java
try{
    ... EIP API Operations ...
}
catch (DKException exc){
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    System.out.println("");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("    Name: " + exc.name());
    System.out.println("    Message: " + exc.getMessage());
    System.out.println(" Message ID: " + exc.getErrorId());
    System.out.println("Error State: " + exc.errorState());
    System.out.println(" Error Code: " + exc.errorCode());
    exc.printStackTrace();
    System.out.println("-----");
} catch (Exception exc) {
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    System.out.println("");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("    Name: " + exc.getClass().getName());
    System.out.println(" Message: " + exc.getMessage());
    exc.printStackTrace();
    System.out.println("-----");
}
```

C++

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

For more information on error detection and handling, see the SConnectDisconnectICM API education sample.

Constants

The constants specified are in the form of DK_CM_ (Common constants) or DK_XX_ (where the XX indicates different content servers). See Table 5 on page 23 for a list of the extensions (extensions appended to each DKDatastore).

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

Java

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

C++

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

Connecting to content servers

An object of the class `DKDatastorexx` (where *xx* indicates a specific content server) represents and manages a connection to a content server, provides transaction support, and runs server commands. See Table 5 for the exact extensions.

If your application terminates abruptly or abnormally, such as when a user enters Ctrl-C against the application, you must ensure that any datastores that are connected get disconnected by your application .

Table 5. Server type and class name terminology

Content server	Class name
DB2 Content Manager Version 8.3	<code>DKDatastoreICM</code>
Earlier IBM Content Manager	<code>DKDatastoreDL</code>
Content Manager OnDemand	<code>DKDatastoreOD</code>
DB2 Content Manager for AS/400 (VisualInfo for AS/400)	<code>DKDatastoreV4</code>
Content Manager ImagePlus for OS/390	<code>DKDatastoreIP</code>
Domino.Doc	<code>DKDatastoreDD</code>
Relational databases	<code>DKDatastoreDB2</code> , <code>DKDatastoreJDBC</code> (for Java) <code>DKDatastoreODBC</code> (for C++)

Establishing a connection

Each `DKDatastorexx` class provides methods for connecting to it and disconnecting from it. The following example uses a DB2 Content Manager library server named `ICMNLSDDB`, the user ID `ICMADMIN` and password `PASSWORD`. For information on Content Manger, see Connecting to the DB2 Content Manager system; for other content servers, see Chapter 8, “Working with other content servers.” The example creates a `DKDatastoreICM` object for the CM content server, connects to it, works with it, then disconnects from it.

Java

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

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

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

For the complete sample application, refer to the SConnectDisconnectICM.java sample.

C++

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

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

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

For the complete sample application, refer to the SConnectDisconnectICM.cpp sample.

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

Connecting and disconnecting from a content server in a client

You use the same code in “Establishing a connection” on page 23 to access a content server from a client application. To do so, simply replace `import com.ibm.mm.sdk.server.*;` with `import com.ibm.mm.sdk.client.*;`. Your client application must handle any communications errors incurred.

Setting and getting content server options

You can access or set the processing options on a content server using the methods in `DKDatastorexx`. The following example shows how to set and get the option for establishing an administrative session on a DB2 Content Manager library server. See the Application Programming Reference for the list of options and their descriptions.

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

Java

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

C++

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

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

Listing content servers

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

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

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

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

Java

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

C++

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

Listing the entities and attributes for a content server

DKDatastorexx 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 Application Programming Reference.

The following example shows how to retrieve the list of item types as well as the list of attributes from a DB2 Content Manager server.

Java

```
. . .
try {
    DKSequentialCollection pCol = null;
    dkIterator pIter = null;
    DKSequentialCollection pCol2 = null;
    dkIterator pIter2 = null;
    DKServerDefICM pSV = null;
    String strServerName = null;
    String strItemType = null;
    DKComponentTypeDefICM itemTypeDef = null;
    DKAttrDefICM attrDef = null;
    DKDatastoreDefICM dsDefICM = null;
    int i = 0;
    int j = 0;
    // ----- Create the datastore and connect (assumes the
    //      parameters for the connection are previously set)
    DKDatastoreICM dsICM = new DKDatastoreICM();
    dsICM.connect(db,userId,pw,"");
    // ----- List the item types
    pCol = (DKSequentialCollection) dsICM.listEntities();
    pIter = pCol.createIterator();
    i = 0;
    while (pIter.more() == true)
    {
        i++;
        itemTypeDef = (DKComponentTypeDefICM)pIter.next();
        strItemType = itemTypeDef.getName();
        System.out.println("item type name [" + i + "] - " + strItemType);
        System.out.println("    type " + itemTypeDef.getType());
        System.out.println("    itemTypeId " + itemTypeDef.getId());
        System.out.println("    compID " + itemTypeDef.getComponentTypeId());
        //continued . . .
    }
}
```

Java (continued)

```
// ----- List the attributes
pCol2 = (DKSequentialCollection) dsICM.listEntityAttrs(strItemType);
pIter2 = pCol2.createIterator();
j = 0;
while (pIter2.more() == true)
{
    j++;
    attrDef = (DKAttrDefICM)pIter2.next();
    System.out.println("Attr name [" + j + "] - " + attrDef.getName());
    System.out.println("    datastoreType " + attrDef.datastoreType());
    System.out.println("    attributeOf " + attrDef.getEntityName());
    System.out.println("    type " + attrDef.getType());
    System.out.println("    size " + attrDef.getSize());
    System.out.println("    id " + attrDef.getId());
    System.out.println("    nullable " + attrDef.isNullable());
    System.out.println("    precision " + attrDef.getPrecision());
    System.out.println("    scale " + attrDef.getScale());
    System.out.println("    stringType " + attrDef.getStringType());
    System.out.println("    sequenceNo " + attrDef.getSequenceNo());
    System.out.println("    userFlag " + attrDef.getUserFlag());
}
dsICM.disconnect();
}
catch(DKException exc)
{
    // ----- Handle the exceptions
```

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

The following C++ example shows how to retrieve the list of index classes and attributes from a DB2 Content Manager server:

C++

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

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


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

C++

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

Working with dynamic data objects (DDOs)

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

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

You use the DKDDO class for dynamic data objects (DDOs) in your IBM DB2 Information Integrator for Content applications. A DKDDO object represents an item, which, for example, could be a DB2 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 DB2 Content Manager, or rows from different tables in a relational database. The DKDDO itself can have properties that apply to the whole DKDDO, instead of to only one particular attribute.

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

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

Creating a DKDDO

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

Java

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

C++

```
DKDDO ddo;
```

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

Java

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

C++

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

Important: In APIs such as DKDatastoreICM and DKDatastoreOD, create a DKDDO by using other constructors (such as the createDDO() method in the DKDatastore XX class). The following example creates a DKDDO by passing in both content server and object type for Content Manager Version 8:

Java

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

For more information about DDO creation, refer to the SItemCreationICM sample.

C++

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

For more information about DDO creation, refer to the SItemCreationICM sample.

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

Java

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

C++

```
// create an earlier DB2 Content Manager datastore
DKDatastoreDL dsDL;
// create a DDO to hold an object type DLSAMPLE in dsDL
DKDDO* cddo = new DKDDO(&dsDL, "DLSAMPLE");
```

Which constructor you use depends on your application; see the Application Programming Reference for information on the constructors.

Adding properties to a DDO

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

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

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

Java

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

C++

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

Creating a persistent identifier (PID)

Each DDO must have a persistent identifier (PID). The PID contains information about the content server's name, type, ID, and object type. The PID identifies the DDO's persistent data location. For example, in earlier DB2 Content Manager content server, the PID is the item ID. The item ID is one of the most important parameters for the retrieve, update, and delete functions. In DB2 Content Manager V8, the PID has five parts (see Chapter 4, "Working with DB2 Content Manager Version 8.3," on page 107 for more information).

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

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

Java

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

C++

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

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

DB2 Content Manager 8.3 connector has a PID class that is a subclass of DKPid. It is called DKPidICM which is the pid used by the DKDDOs and dkResources.

Working with data items and properties

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

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

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

Java

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

// Set attributes (Additional attributes set in SItemCreationICM sample)
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar"),
    "abcdefg");
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_integer"),
    new Integer("123"));
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_short"),
    new Short("5"));
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_time"),
    java.sql.Time.valueOf("10:00:00"));

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

This example was taken SItemCreationICM sample.

C++

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

// Set attributes (Additional attributes set in SItemCreationICM sample)
// NOTE: Values below are automatically converted to type DKAny
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_varchar"),
    DKString("this is a string value"));
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_integer"),
    (long) 123);
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_short"),
    (short) 5);
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_time"),
    DKTime("10.00.00"));

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

This example was taken from the SItemCreationICM sample.

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

The following example accesses attribute values of a DDO.

Java

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

```
String attrVal1 = (String) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar"));  
Integer attrVal2 = (Integer) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_integer"));  
Short attrVal3 = (Short) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_short"));  
Time attrVal4 = (Time) ddo.getData(ddo.dataId(  
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_time"));  
  
System.out.println("Attr 'S_varchar' value: "+attrVal1);  
System.out.println("Attr 'S_integer' value: "+attrVal2);  
System.out.println("Attr 'S_short' value: "+attrVal3);  
System.out.println("Attr 'S_time' value: "+attrVal4);
```

This example was taken from the SItemRetrievalICM sample.

C++

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

```
DKString attrVal1 = ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,  
    DKString("S_varchar"))).toString();  
long attrVal2 = (long) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,  
    DKString("S_integer")));  
short attrVal3 = (short) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,  
    DKString("S_short")));  
DKTimestamp attrVal4 = (DKTimestamp) ddo->getData(ddo->dataId(  
    DK_CM_NAMESPACE_ATTR, DKString("S_time")));  
  
cout << "Attr 'S_varchar' value: " << attrVal1 << endl;  
cout << "Attr 'S_integer' value: " << attrVal2 << endl;  
cout << "Attr 'S_short' value: " << attrVal3 << endl;  
cout << "Attr 'S_time' value: " << attrVal4 << endl;
```

This example was taken from the SItemRetrievalICM sample.

Getting the DKDDO and attribute properties

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

Java

```
short prop_id = ddo.propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    short type = ((Short) ddo.getProperty(prop_id)).shortValue();
    switch(type) {
        case DK_CM_DOCUMENT:
            // --- process a document
            ....
            break;
        case DK_CM_FOLDER:
            // --- process a folder
        case DK_CM_ITEM:
            // --- Process an item in Content Manager
            ....
            break;
    }
}
```

For more information on accessing item type properties, refer to the SItemRetrievalICM API Education Sample.

C++

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

For more information on accessing item type properties, refer to the SItemRetrievalICM API Education Sample.

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

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

Java

```
data_id = ddo.dataId("Title"); // get data_id of Title
// ----- Get the number of properties for the attribute
short number_of_data_prop = ddo.dataPropertyCount(data_id);
// ----- Display all data properties belonging to this attribute
//         using a loop; the index starts at 1
for(short i = 1; i <= number_of_data_prop; i++) {
    System.out.println(i + " Property Name = " +
        ddo.getDataPropertyName(data_id,i)
        + " value = " + ddo.getDataProperty(data_id,i));
}
```

For the complete sample application, refer to the SItemRetrievalICM sample.

C++

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

For the complete sample application, refer to the SItemRetrievalICM sample.

Displaying the whole DDO

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

Java

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

For a complete example of accessing and printing a DDO (and all subcomponents), refer to `SItemRetrievalICM` and its `printDDO()` static function.

C++

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

For a complete example of accessing and printing a DDO (and all subcomponents), refer to `SItemRetrievalICM` and its `printDDO()` static function.

Deleting a DDO (C++ only)

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

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

Working with extended data objects (XDOs)

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

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

Java

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

DB2 Content Manager only: Since CM 8 requires XDOs to be of the correct subclass, DKDDOs should always be created using the `DKDatastoreICM's createDDO()` methods. This allows the system to automatically set up important information in the DKDDO structure, and provides greater functionality such as resources, CM document model, and folders. Items of type resource (returned from `DKDatastoreICM.createDDO()`) can be cast to the correct XDO or subclass depending on the XDO classification. For more information on creating items in general and the `DKDatastoreICM.createDDO()` function, see the `SItemCreationICM` sample.

To create an XDO for binary objects use `DKBlobxx`, where `xx` is the suffix representing the specific server. For example, use `DKBlobICM` for DB2 Content Manager, `DKBlobOD` for OnDemand, or `DKBlobIP` for ImagePlus for OS/390. When you create a `DKBlobxx` object, you must pass it the content server `DKDatastorexx`. For Content Manager, you use `DKLobICM` to create the XDO.

Using an XDO persistent identifier (PID)

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

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

Understanding XDO properties

Use the methods of the `DKBlobxx` to set the properties of an XDO where they apply; all properties are not available for all content servers. When loading, default values for the properties are set if specific values are not specified. For example, the following defaults are use with earlier DB2 Content Manager:

RepType (representation type)

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

ContentClass

The default is DK_CM_CC_UNKNOWN. For the valid values, see DKConstant2DL.h in the \ Program Files\IBM\B2CMV8 \include directory for Information Integrator for Content.

AffiliatedType

The default is: DK_DL_BASE.

AffiliatedData

The default is: NULL.

To index object content with earlier DB2 Content Manager correctly, you must set SearchEngine, SearchIndex, and SearchInfo in the extension object DKSearchEngineInfoDL.

For working with XDOs in Content Manager, see “Working with items” on page 122.

C++ Tip: For the valid values of ContentClass, See the file INCLUDE/DKConstant2DL.h provided with DB2 Content Manager.

DB2 and ODBC configuration strings (C++ only)

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

CC2MIMEFILE=(filename)

Specify the cmbcc2mime.ini file (optional).

DSNAME=(content server name)

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

AUTOCOMMIT=ON | OFF

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

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

NATIVECONNECTSTRING=(native connect string)

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

SCHEMA=name

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

Java programming tips

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

For earlier Content Manager, DB2 Content Manager for AS/400, and IP 390, you identify an XDO by the combination of item ID, part ID and the RepType. For RDB, the key to identify an XDO is combination of table, column and data predicate string. To handle a stand-alone XDO, you provide the item ID and part ID. The RepType is optional since the system provides a default value for it.

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

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

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

Java

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

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

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

C++ programming tips

For DB2 Content Manager, VI400 and IP390, you identify an XDO by the combination of item ID, part ID, and RepType. For Relational Databases, the combination of table name, column name and datapredicate is the key to identify an XDO. For a standalone XDO, you must provide the item ID and part ID. RepType is optional, because the system provides a default value (FRN\$NULL).

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

Important: When adding a part for the search manager to index on a DB2 Content Manager content server, you must have a valid part ID and cannot set the part ID to 0.

Programming an XDO as a part of DDO

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

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

Java

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

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

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

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

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

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

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

For the complete sample application, refer to the `SDocModelItemICM.java` sample. `SResourceItemCreationICM` shows more examples of XDO use.

C++

```
// Create a document
DKDDO* ddoDocument = dsICM->createDDO("S_docModel", DK_CM_DOCUMENT);
// Create Parts
DKLobICM* base = (DKLobICM*) dsICM->createDDO("ICMBASE",
DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM* baseText1 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
DK_ICM_SEMANTIC_TYPE_Basetext);
DKTextICM* baseText2 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
DK_ICM_SEMANTIC_TYPE_Basetext);

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

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

For the complete sample application, refer to the SDocModelItemICM.java sample. SResourceItemCreationICM shows more examples of XDO use.

Programming a stand-alone XDO

All of the following examples are specific to DB2 Content Manager 8.2. For examples for earlier Content Manager and other content servers, see “Representing items using DDOs” on page 120, Chapter 8, “Working with other content servers,” on page 259, and see the sample programs in the samples directory.

Adding an XDO from the buffer

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

Java

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

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

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

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

For the complete sample application, refer to the SResourceItemCreationICM sample.

C++

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

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

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

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

For the complete sample application, refer to the SResourceItemCreationICM sample.

Adding an XDO from a file

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

Java

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

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

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

For the complete sample application, refer to the SResourceItemCreationICM sample.

C++

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

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

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

For the complete sample application, refer to the SResourceItemCreationICM sample.

Adding an annotation object to an XDO

The following example adds an annotation part to a document in Content Manager.

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

Java

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

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

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

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

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

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

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

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

For the complete sample application, refer to the SDocModelItemICM sample.

C++

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

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

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

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

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

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

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

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

For the complete sample application, refer to the SDocModelItemICM sample.

Examples of working with an XDO

The following examples illustrate using a stand-alone XDO.

Retrieving, updating, and deleting an XDO

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

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

Java

```
// Given: String pidString

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

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

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

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

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

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

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

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

This code sample comes from SResourceItemRetrievalICM, SResourceItemUpdateICM, and SResourceItemDeletionICM sample.

C++

```
// Given: DKString pidString

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

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

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

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

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

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

// Delete item
lob->del();

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

This code sample comes from SResourceItemRetrievalICM, SResourceItemUpdateICM, and SResourceItemDeletionICM sample.

Invoking an XDO function

This example demonstrates how to test the DKBlob class using an earlier DB2 Content Manager server. For this example, you must know the item ID and part ID of the XDO.

Java

```
public class txdomiscDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int    partId = 5;
        String itemId = "GAWCVGGVFUG428UJ";
        String repType = "";
        // Check the number of arguments for main and determine what to do
        if (args.length == 3)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            itemId = args[2];
            System.out.println("You enter: java txdomiscDL " +
                + partId + " " + repType + " " + itemId);
        }
        if (args.length == 2)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            System.out.println("You enter: java txdomiscDL " +
                + partId + " " + repType);
        }
        if (args.length == 1)
        {
            partId =(short)Integer.parseInt(args[0], 10);
            System.out.println("You enter: java txdomiscDL " + partId );
            System.out.println("The supplied default repType = " + repType);
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
            System.out.println("invoke: java txdomiscDL  ");
            System.out.println("No parameter, following defaults provided:");
            System.out.println("    default partId = " + partId);
            System.out.println("    default repType = " + repType);
            System.out.println("    default itemId = " + itemId);
        }

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

            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            apid.setRepType(repType);
            axdo.setPidObject(apid);
            System.out.println("repType=" + apid.getRepType());
            System.out.println("itemid=" + apid.getItemId());
            System.out.println("partId=" + apid.getPartId());
        }
        // continued...
    }
}
```

Java (continued)

```
// ----- Before retrieve
System.out.println("before retrieve:");
System.out.println("  contentclass=" + axdo.getContentClass());
System.out.print("  content length=" + axdo.length());
System.out.println(" (the length of this object instance - in memory)");
System.out.print("  getSize=" + axdo.getSize());
System.out.println(" (get the object size without retrieving object)");
System.out.println("  createdTimestamp=" + axdo.getCreatedTimestamp());
System.out.println("  updatedTimestamp=" + axdo.getUpdatedTimestamp());
axdo.retrieve();

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

}
catch (DKException exc)
{
    // ----- Handle the exceptions
}
```

C++

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

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

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

C++ (continued)

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

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

//== after retrieve
cout<<"after retrieve:"<<endl;
cout<<" content class="<<axdo->getContentClass()<<endl;
cout<<" content length="<<axdo->length();
cout<<" (the length of this object instance - in memory)"<<endl;
cout<<" getSize="<<axdo->getSize();
cout<<" (get the object size without retrieving object)"<<endl;
cout<<" createdTimestamp="<<axdo->getCreatedTimestamp()<<endl;
cout<<" updatedTimestamp="<<axdo->getUpdatedTimestamp()<<endl;
cout<<" mimeType="<<axdo->getMimeType()<<endl;
int atype = axdo->getAffiliatedType();
cout<<" affiliatedType= "<<axdo->getAffiliatedType()<<endl;
if (atype == DK_DL_ANNOTATION)
{
    DKAnnotationDL* ann=(DKAnnotationDL*)axdo
    ->getExtension("DKAnnotationDL");
    cout <<" pageNumber= "<<ann->getPageNumber()<<endl;
    cout <<" partId= "<<ann->getPart()<<endl;
    cout <<" X="<<ann->getX()<<endl;
    cout <<" Y="<<ann->getY()<<endl;
}
//== open content
int concls = axdo->getContentClass();
if (concls == DK_DL_CC_ASCII)
    axdo->setInstanceOpenHandler("notepad", TRUE);
else if (concls == DK_DL_CC_GIF)
    axdo->setInstanceOpenHandler("lviewpro", TRUE);
else if (concls == DK_DL_CC_AVI)
    axdo->setInstanceOpenHandler("mplay32", TRUE);
axdo->open();
// continued...
```

C++ (continued)

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

Adding an XDO media object in earlier DB2 Content Manager

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

Java

```
public class txdoAddVSDL implements DKConstantDL
{
    // ----- Main method
    public static void main(String[] args)
    {
        String fileName = "/icing1.mpg1";           //a media object
        String itemId = "K1A04EWBVHJAV1D7";        //a known itemId
        int partId = 45;
        // ----- Check the arguments for main
        if (args.length == 3)
        {
            fileName = args[0];
            partId = (int)Integer.parseInt(args[1], 10);
            itemId = args[2];
            System.out.println("You enter: java txdoAddVSDL " +
                fileName + " " + partId + " " + itemId);
        }
        if (args.length == 2)
        {
            fileName = args[0];
            partId = (int)Integer.parseInt(args[1], 10);
            System.out.println("You enter: java txdoAddVSDL " +
                fileName + " " + partId );
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 1)
        {
            fileName = args[0];
            System.out.println("You enter: java txdoAddVSDL " + fileName);
            System.out.println("The supplied default partId = " + partId);
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
            System.out.println("invoke: java txdoAddVSDL <filename> <part ID> <item ID>");
            System.out.println("No parameter, following defaults will be provided:");
            System.out.println("    default fileName = " + fileName);
            System.out.println("    default partId = " + partId);
            System.out.println("    default itemId = " + itemId);
        }
        // ----- Processing
        try
        {
            // ----- connect to datastore
            DKDatastoreDL dsDL = new DKDatastoreDL();
            // replace following with your library server, userid, password
            System.out.println("connecting to datastore...");
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            System.out.println("datastore connected");
            // ----- create xdo and pid
            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            axdo.setPidObject(apid);
            // you must use the content class DK_DL_CC_IBMVSS for a media object
            axdo.setContentClass(DK_DL_CC_IBMVSS);
            System.out.println("contentClass=" + axdo.getContentClass());
            System.out.println("partId = " + axdo.getPartId());
        }
        // continued...
    }
}
```

Java (continued)

```
// ----- set up DKMediaStreamInfoDL
DKMediaStreamInfoDL aVS = new DKMediaStreamInfoDL();
aVS.setMediaFullFileName(fileName);
// if fileName contain a list of media segments then use following
//      aVS.setMediaObjectOption(DK_VS_LIST_OF_OBJECT_SEGMENTS);
aVS.setMediaObjectOption(DK_DL_VS_SINGLE_OBJECT);
aVS.setMediaHostName("<insert hostname here>");
aVS.setMediaUserId("<insert user ID here>");
aVS.setMediaPassword("<insert password here>");
// following are optional, if not set default value will be provided
aVS.setMediaNumberOfUsers(2);
aVS.setMediaAssetGroup("AG");
// ----- same as defined in VideoCharger server
aVS.setMediaType("MPEG1");
aVS.setMediaResolution("SIF");
aVS.setMediaStandard("NTSC");
aVS.setMediaFormat("SYSTEM");
axdo.setExtension("DKMediaStreamInfoDL", (dkExtension)aVS);
System.out.println("about to call add()");
axdo.add();
System.out.println("add successfully.....");
System.out.println("after added check for status:");
boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
}
dsDL.disconnect();
dsDL.destroy();
}
catch (DKException exc) {
    try {
        dsDL.destroy();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    System.out.println("Exception name " + exc.name());
    System.out.println("Exception message " + exc.getMessage());
    exc.printStackTrace();
}
catch (Exception exc){
    try {
        dsDL.destroy();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    System.out.println("Exception message " + exc.getMessage());
    exc.printStackTrace();
}
}
```

C++

```
void main(int argc, char *argv[])
{
    DKString itemId, fileName;
    int partId;
    itemId = "K1A04EWBVHJAV1D7";
    partId = 22;
    fileName = "/icing1.mpg1";
    if (argc == 1)
    {
        cout<<"invoke: txdoAddVSDL <fileName> <partId> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default fileName = "<<fileName<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        fileName = DKString(argv[1]);
        cout<<"you enter: txdoAddVSDL "<<argv[1]<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        fileName = DKString(argv[1]);
        partId = atoi(argv[2]);
        cout<<"you enter: txdoAddVSDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        fileName = DKString(argv[1]);
        partId = atoi(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"enter: txdoAddVSDL "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    try
    {
        // connect to datastore
        cout << "Connecting datastore ..." << endl;
        DKDatastoreDL dsDL;
        // replace following with your library server, user ID, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;
        // *** create xdo and pid
        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
        axdo ->setPidObject(apid);
        // you must use the content class DK_DL_CC_IBMVSS for a media object
        axdo ->setContentClass(DK_DL_CC_IBMVSS);
        cout <<"itemId= "<<axdo->getItemId()<<endl;
        cout <<"partId= "<<axdo->getPartId()<<endl;
        cout <<"repType= "<<axdo->getRepType()<<endl;
        cout <<"content class="<< axdo->getContentClass()<<endl;
    }
    // continued...
```

C++ (continued)

```
// *** setup DKMediaStreamInfoDL
DKMediaStreamInfoDL aVS;
aVS.setMediaFullFileName(fileName);
aVS.setMediaObjectOption(DK_DL_VS_SINGLE_OBJECT);
aVS.setMediaHostName("<insert hostname here>");
aVS.setMediaUserId("<insert user ID here>");
aVS.setMediaPassword("<insert password here>");

//following are optional, if not set then default value will be provided
aVS.setMediaNumberOfUsers(1);
aVS.setMediaAssetGroup("AG");
// *** same as defined in VideoCharger server
aVS.setMediaType("MPEG1");
aVS.setMediaResolution("SIF");
aVS.setMediaStandard("NTSC");
aVS.setMediaFormat("SYSTEM");

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

cout<<"after added check for status:"<<endl;
DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
if (flag2)
{
    DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
        axdo->getExtension("DKMediaStreamInfoDL");
    cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
    cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
    cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
    cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
    cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
    cout<<" MediaState(dynamic)="<<
        axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;
}
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0;g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

Deleting an XDO media object

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

Java

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

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

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

            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            apid.setRepType(repType);
            axdo.setPidObject(apid);
            boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
            System.out.println("isMediaObject?=" + flag2);
        }
        // continued...
```

Java (continued)

```
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
    // ----- set delete option for media object
    axdo.setOption(DK_DL_OPT_DELETE_OPTION,
        (Object)new Integer(DK_DL_DELETE_NO_DROPITEM_MEDIA_AVAIL));
    System.out.println("The delete option =" +
        (Integer)(axdo.getOption(DK_OPT_DL_DELETE_OPTION)));
}

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

C++

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

    try
    {
        cout << "Connecting datastore ..." << endl;
        // replace following with your library server, user ID, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
        apid ->setRepType(repType);
        axdo ->setPidObject(apid);
        cout <<"itemId= "<<axdo->getItemId()<<endl;
        cout <<"partId= "<<((DKPidXDODL*)(axdo->getPidObject()))
            ->getPartId()<<endl;
        DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
        cout <<"isMediaObject? = "<<flag2<<endl;
    }
    // continued...
```


C++ (continued)

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

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

}
cout<<"about to do del()"<<endl;
axdo->del();
cout<<"del successfully..."<<endl;
flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
cout<<"after delete isMediaObject? = "<<flag2<<endl;
delete axdo;
delete apid;
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0;g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

Retrieving an XDO media object

The following example shows how to retrieve an XDO media object. The retrieved object contains only the media metadata, not the media object itself. For this example you must know the item ID and part ID of the XDO.

Java

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

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

Java (continued)

```
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
}
System.out.println("before retrieve.....");
System.out.println(" lob length=" + axdo.length());
System.out.println(" size=" + axdo.getSize());
System.out.println(" createdTimestamp="+axdo.getCreatedTimestamp());
System.out.println(" updatedTimestamp="+axdo.getUpdatedTimestamp());
// ----- Perform the retrieve call
axdo.retrieve();

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

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

C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "K1A04EWBVHJAV1D7";
    partId = 1;
    repType = "FRN$NULL";
    if (argc == 1)
    {
        cout<<"invoke: txdoRetxsDL <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        itemId = DKString(argv[3]);
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]
            <<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    try
    {
        cout << "Connecting datastore ..." << endl;
        // replace following with your library server, userid, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;

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

C++ (continued)

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

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

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

C++ (continued)

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

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

Adding an XDO to a storage collection

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

The following example uses `DKStorageManageInfoDL`, for an earlier DB2 Content Manager server; for Content Manager Version 8 and later, see Chapter 11, “Working with XML services (Java only),” on page 429.

Java

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

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

dsDL.disconnect(); // disconnect from and destroy datastore
dsDL.destroy();
// ----- Handle the exceptions
```

C++

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

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

dsDL.disconnect(); //disconnect from datastore
System.out.println("datastore disconnected");
```

See the following code samples in the samples directory for examples of adding search indexed objects and media objects to Content Manager.

- TxdoAddBsmsDL
- TxdosAddBsmsDL

- TxdoAddFsmsDL
- TxdosAddFsmsDL
- TxdomAddsmsDL

Changing the storage collection of an XDO

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

Java

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

C++

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

Creating documents and using the DKPARTS attribute

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

DB2 Content Manager only: A document is an item that can be stored, retrieved, and exchanged among Content Manager systems and users as a separate unit. An item given the *document* semantic type is expected to contain information that forms a document, but does not rigidly mean an implementation of a specific document model. An item created from a document (also known as a document model) classified item type means that the item will contain document parts, a specific implementation of a document model provided by Content Manager. Document classified item types can create items given either the document or folder semantic type. The document parts can include varied types of content, including for example, text, images, and spreadsheets.

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

Java

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

// Create parts
DKLobICM base = (DKLobICM) dsICM.createDDO("ICMBASE",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM baseText1 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASETEXT);
DKTextICM baseText2 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASETEXT);

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

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

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

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

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

For information on creating documents with parts, refer to the SDocModelItemICM API Education Sample.

C++

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

// Create Parts
DKLobICM* base = (DKLobICM*) dsICM->createDDO("ICMBASE",
DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM* baseText1 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
DK_ICM_SEMANTIC_TYPE_BASSETEXT);
DKTextICM* baseText2 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
DK_ICM_SEMANTIC_TYPE_BASSETEXT);

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

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

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

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

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

For information on creating documents with parts, refer to the SDocModelItemICM API Education Sample.

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

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

Java

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

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

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

For the complete sample application, refer to SDocModelItemICM sample.

C++

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

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

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

For information on creating documents with parts, refer to the SDocModelItemICM API Education Sample.

Creating folders and using the DKFOLDER attribute

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

DB2 Content Manager only: A folder is an item of any item type, regardless of classification, with the *folder* semantic type. Any item with the folder semantic type will contain specific folder functionality provided by Content Manager, in addition to all non-resource item capabilities and any additional functionality available from an item type classification such as document model or resource. Folders may contain any number of items of any type, including documents and subfolders. A folder is indexed by attributes.

The following example creates a folder and adds contents in Content Manager.

Requirement: The user-defined item type, S_simple, must be defined in the system, as defined in the SItemTypeCreationICM API education sample.

Java

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

// Create and save contents to place in folder
DKDDO ddoDocument = dsICM.createDDO("S_simple", DKConstant.DK_CM_DOCUMENT);
DKDDO ddoFolder2 = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
DKDDO ddoItem = dsICM.createDDO("S_simple", DKConstant.DK_CM_ITEM);
ddoDocument.add();
ddoItem.add();
ddoFolder2.add();

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

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

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

For more information on creating Folders, refer to the SFolderICM API Education Sample.

C++

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

// Create and save contents to place in folder
DKDDO* ddoDocument = dsICM->createDDO("S_simple", DK_CM_DOCUMENT);
DKDDO* ddoFolder2 = dsICM->createDDO("S_simple", DK_CM_FOLDER);
DKDDO* ddoItem = dsICM->createDDO("S_simple", DK_CM_ITEM);
ddoDocument->add();
ddoItem->add();
ddoFolder2->add();

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

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

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

For more information on creating Folders, refer to the SFolderICM API Education Sample.

In DB2 Content Manager Version 8, the folder item does not own the folder contents. An item may be added to multiple folders. Removing an item from a folder simply breaks the folder-content relationship managed by the system. Items in the folder must be updated and deleted independently. In earlier versions of Content Manager, the DDO owns the contents in the collection.

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

Java

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

// Get the DKFolder object.
short dataid = folder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
                             DKConstant.DK_CM_DKFOLDER);

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

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

For the complete sample application, refer to the SFolderICM education sample.

C++

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

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

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

For the complete sample application, refer to the SFolderICM education sample.

Using DKAny (C++ only)

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

- null
- (unsigned) short

- short
- (unsigned) long
- long
- float
- double
- char
- TypeCode
- DKBoolean
- DKString
- DKDate
- DKTime
- DKTimestamp
- DKByteArray
- DKDecimal
- DKNVPair

A DKAny can only be NULL if it has not been assigned a value or the `DKAny::setNull()` method has been called (DKAny 1969 any). After it has been assigned a value, `DKAny::isNull()` returns FALSE.

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

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

Using type code

You can determine the current type of a DKAny object by calling the `typeCode` function, which returns a `TypeCode` object, that is, `tc_null` for null, `tc_short` for short, and so forth. See the Application Programming Reference for a complete listing of type codes.

Managing memory in DKAny

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

Using constructors

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

C++

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

Getting the type code

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

C++

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

Assigning a new value to DKAny

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

C++

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

Assigning a value from DKAny

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

C++

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

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

C++

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

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

C++

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

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

C++

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

Displaying DKAny

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

C++

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

Destroying DKAny

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

C++

```
DKDDO* ddo = new DKDDO;           // creates a DKDDO in the heap
DKAny anyA((dkDataObjectBase*)ddo);
DKAny* anyB = new DKAny(anyA);     // creates anyB in the heap
                                   // anyA and anyB contains a
                                   // reference to the same ddo

...
delete anyB;                       // delete anyB, does not delete ddo
if (anyA.typeCode() == DKAny::tc_dobase)
    delete ((dkDataObjectBase*) anyA.value()); // deletes the ddo
```

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

Programming tips

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

C++

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

Using collections and iterators

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

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

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

Using sequential collection methods

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

Java

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

C++

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

Using the sequential iterator

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

Java

`dkIterator`, the base iterator, supports the `next`, `more`, and `reset` methods. The subclass `DKSequentialIterator` contains more methods. You create an iterator by calling the `createIterator` method on the collection. After creating the iterator you may use the methods below to traverse the collection. The following example shows how to use an iterator:

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

C++

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

```
dkIterator* iter = sq.createIterator(); // create an iterator for sq  
DKAny* member;  
                                     // while there are more members  
                                     // get the current member and  
                                     // advance iter to the next member  
while(iter->more()) {  
    member = iter->next();  
  
    cout << *member << endl;           // display it, if you want to  
    ...                               // do other processing  
}  
delete iter;                         // do not forget to delete iter
```

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

Java

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

C++

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

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

Tip: When you remove the current member, the iterator is advanced to the next member. When removing a member inside a loop, check it as in the following example. You should create a new iterator when an item is deleted from the iterator.

Java

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

C++

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

Managing memory in collections (C++ only)

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

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

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

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

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

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

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

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

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

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

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

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

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

Sorting the collection

Use the `sort` function to sort collection members in either ascending or descending order based on a specified key. You must pass a sort object and the desired order. The interface for sort objects is defined in `dkSort.java` (Java) or `dkSort.hpp` (C++).

You can write your own sort function for sorting your specific collection. The following example illustrates how to sort a collection of DDOs:

Java

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

C++

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

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

Understanding federated collection and iterator

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

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

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

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

Restriction: You cannot query a federated collection.

Figure 6 on page 81 shows the structure and behavior of DKFederatedCollection.

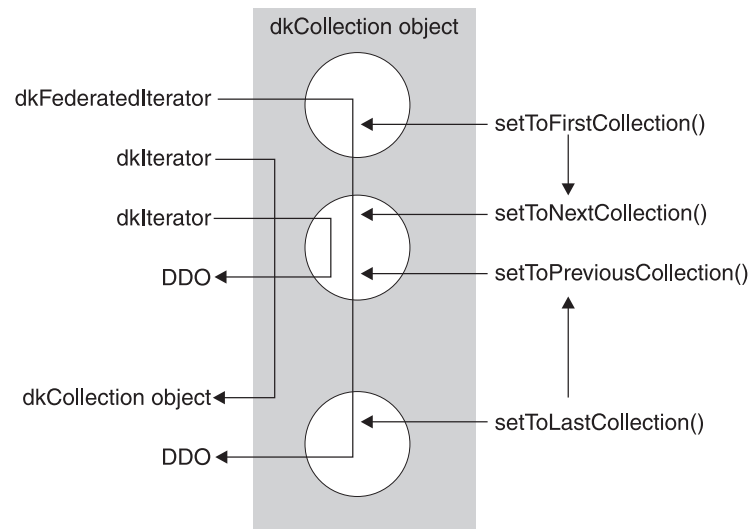


Figure 6. Structure and behavior of DKFederatedCollection

In Figure 6, the rectangle represents the DKFederatedCollection containing several smaller circles which are DKResults objects. The dkFederatedIterator traverses collection boundaries and returns a DDO each time.

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

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

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

Querying a content server

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

For the content servers that support query objects, there are four types of query objects: parametric, text, image and combined. The combined query is composed of both text and parametric queries. Not all content servers can perform combined queries. Earlier DB2 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 DB2 Content Manager server” on page 187. For information about querying an earlier DB2 Content Manager server, see Chapter 8, “Working with other content servers,” on page 259.

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

`dkResultSetCursor` can also be used to rerun a query by calling the `close` and `open` methods. This is described in “Using the result set cursor” on page 99.

`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 100 for more information.

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

Differences between `dkResultSetCursor` and `DKResults`

A `dkResultSetCursor` and a `DKResults` collection have the following differences:

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

Restriction: You cannot rerun a query on a Domino.Doc content server even when using a `dkResultSetCursor`.

- The `DKResults` contains the entire result set and supports a bi-directional iterator.
- Leaving a `dkResultSetCursor` open for long periods of time may degrade performance of concurrent users for some content servers.

Using parametric queries

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

Note: The query examples in the following sections apply to earlier Content Manager. For query information about DB2 Content Manager V8, see *Querying the DB2 Content Manager server*, in *Working with DB2 Content Manager 8.2*.

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 DB2 Content Manager. For examples of query string in Content Manager, see “Example searches using the query language” on page 196. The condition of the query is to search for all documents or folders where the attribute

DLSEARCH_DocType is not null. The maximum number of results returned is limited to five, and the content is set to YES so that contents of the document or folder are returned.

Java

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

C++

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

The example specifies that an earlier DB2 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. See the information on the content server you want to search or the Application Programming Reference for more information. If the attribute name has more than one word or is in a DBCS language, it should be enclosed in apostrophes ('). If the attribute value is in DBCS, it should be enclosed in double quotation marks (").

Formulating a parametric query on multiple criteria

You can specify more than one search criteria for a parametric query. The following example shows how to specify a query on two index classes for earlier DB2 Content Manager:

Java

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS1,MAX_RESULTS=3," +  
             "COND=(DLSEARCH_DocType <> null);" +  
             "INDEX_CLASS=GP2DLS1,MAX_RESULTS=8," +  
             "COND=('First name'==\"Robert\"));" +  
             "OPTION=(CONTENT=YES;" +  
             "TYPE_QUERY=DYNAMIC;" +  
             "TYPE_FILTER=FOLDERDOC)";
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,MAX_RESULTS=3,";  
cmd += "COND=(DLSEARCH_DocType <> NULL);";  
cmd += "INDEX_CLASS=DLSAMPLE,MAX_RESULTS=8,";  
cmd += "COND=('First name' == \"Robert\"));";  
cmd += "OPTION=(CONTENT=YES,";  
cmd += "TYPE_QUERY=DYNAMIC,";  
cmd += "TYPE_FILTER=FOLDERDOC)";
```


Executing a parametric query

After you have a query string you create the query object. The `DKDatastorexx` that represents a content server contains a method for creating a query object. You use the query object to execute the query and obtain the results. The following example shows how to create a parametric query object and execute the query on an earlier DB2 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.

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

Java

```
// ----- Create the datastore, the query object, and the results set
DKDatastoreDL dsDL = new DKDatastoreDL();
dkQuery pQry = null;
DKResults pResults = null;
DKNameValuePair parms[] = null;
// ----- Connect to the datastore
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
             "MAX_RESULTS=5," +
             "COND=(DLSEARCH_DocType <> NULL));" +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=STATIC;" +
             "TYPE_FILTER=FOLDERDOC)";

// ----- Create the query using the query string
pQry = dsDL.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Execute the query
pQry.execute(parms);
// ----- Process the results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ----- Disconnect when you are through
dsDL.disconnect();
dsDL.destroy();
```

This example was taken from the `TSamplePQryDL.java` sample.

C++

```
DKDatastoreDL dsDL;
dkQuery* pQry;
DKAny any;
DKResults* pResults;

cout << "connecting to datastore" << endl;
dsDL.connect(libsrv,userid,pw);
cout << "datastore connected libsrv: "<<libsrv<<" userid: "<<userid<<endl;

DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,";
cmd += "MAX_RESULTS=5,";
cmd += "COND=(DLSEARCH DocType <> NULL));";
cmd += "OPTION=(CONTENT=YES);";
cmd += "TYPE_QUERY=STATIC;TYPE_FILTER=FOLDERDOC>";
cout << "query string " << cmd << endl;
cout << "create query" << endl;
pQry = dsDL.createQuery(cmd);
cout << "executing query" << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "get query results" << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsDL.disconnect();
```

This example was taken from the TSAMPLEPQryDL.cpp sample.

Executing a parametric query from a content server

The DKDatastorexx that represents a content server has a method to execute a query. The following example shows how to execute a parametric query on an earlier DB2 Content Manager content server. After the query is executed, the results are returned in a dkResultSetCursor object.

Java

```
// ----- Create the datastore and cursor
DKDatastoreDL dsDL = new DKDatastoreDL();
dkResultSetCursor pCur = null;
DKNameValuePair parms[] = null;
// ----- Connect to the content server
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
             "MAX_RESULTS=5," +
             "COND=((DLSEARCH_DocType <> NULL)" +
             "AND (DLSEARCH_Date >= 1995)))"; +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC)";

...
// ----- Execute the query using the query string
pCur = dsDL.execute(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Process query results as you want
...
// ----- When finished with the cursor, delete it, and disconnect
pCur.destroy();
dsDL.disconnect();
dsDL.destroy();
```

This example was taken from the TExecuteDL.java sample.

C++

```
...
DKDatastoreDL dsDL;
dkResultSetCursor* pCur = 0;
cout << "Datastore DL created" << endl;
cout << "connecting to datastore" << endl;
dsDL.connect(libsrv,userid,pw);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;
// DKString cmd = "SEARCH=(COND=('DLSEARCH_DocType' == \"html\"));";
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,";
cmd += "MAX_RESULTS=5,";
cmd += "COND=(DLSEARCH_DocType <> NULL));";
cmd += "OPTION=(CONTENT=YES;";
cmd += "TYPE_QUERY=STATIC;TYPE_FILTER=FOLDERDOC)";
cout << "query string " << cmd << endl;
cout << "executing query" << endl;
pCur = dsDL.execute(cmd);
cout << "query executed" << endl;
...
...
if (pCur != 0)
delete pCur;
dsDL.disconnect();
...
```

This example was taken from the TExecuteDL.cpp sample.

Evaluating a parametric query from a content server

The DKDatastorexx that represents a content server has a method to evaluate a query. The results are returned in a DKResults collection. The following example shows how to evaluate a parametric query on an earlier DB2 Content Manager content server.

Java

```
// ----- Create the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
             "COND=((DLSEARCH_Date >= \"1995\") AND " +
             "      (DLSEARCH_Date <= \"1996\")))";
             "OPTION=(CONTENT=NO;" +
             "      TYPE_QUERY=DYNAMIC;" +
             "      TYPE_FILTER=FOLDERDOC)";

DKNameValuePair parms[] = null;
DKDDO item = null;
// ----- Create the datastore and connect
// replace following with your library server, user ID,
// password DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

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

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS5,";
cmd += "COND=((DLSEARCH_Date >= \"1995\") AND ";
cmd += "(DLSEARCH_Date <= \"1996\")))";
cmd += "OPTION=(CONTENT=NO;";
cmd += "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)";

...
DKAny any = dsDL.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
    element = pIter->next();
    item = (DKDDO*)element->value();
    // Process the DKDDO
}
delete pIter;
delete pResults;
dsDL.disconnect();
```

Using text query

In Content Manager Version 8 and later, text and parametric queries are integrated; see “Creating combined parametric and text search” on page 194.

In earlier DB2 Content Manager, you can perform text and parametric searches. Text searches query the text indexes created by the DB2 Text Information Extender to search the actual document text.

Formulating a text query string

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

Java

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

C++

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

Formulating a text query on multiple indexes

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

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

Java

```
DKString cmd = "SEARCH=(COND=(UNIX OR member));";  
cmd += "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2); MAX_RESULTS=5)";
```

C++

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

Running a text query

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

Java

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

This example was taken from the TSampleTQryTS.java sample.

C++

```
DKDatastoreTS dsTS;
dkQuery* pQry;
DKAny any;
DKResults* pResults;

cout << "connecting to datastore" << endl;
//dsTS.connect("zebra","7502",DK_CTYP_TCPIP);
dsTS.connect(srchSrv,"","");
cout << "connected to datastore srchSrv: " << srchSrv << endl;

DKString cmd = "SEARCH=";
cmd += "(COND=(UNIX OR member));";
cmd += "OPTION=(SEARCH_INDEX=";
cmd += srchIndex;
cmd += ")";
cout << "query string " << cmd << endl;
cout << "create query" << endl;
pQry = dsTS.createQuery(cmd);
cout << "executing query" << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "get query results" << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsTS.disconnect();
```

This example was taken from the TSampleTQryTS.cpp sample.

Running a text query from a content server

The DKDatastorexx used to represent a content server provides a method to run a query. The results are returned in a dkResultSetCursor object. The following example shows how to run a text query against an earlier DB2 Content Manager server:

Java

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

// ----- Formulate the query string
String cmd = "SEARCH=(COND=(internet OR UNIX));" +
             "OPTION=(SEARCH_INDEX=TMINDEX);" +
             "MAX_RESULTS=5";

...
// ----- Execute the query and process the results
as appropriate
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);
...
// ----- When finished, delete the cursor and disconnect
pCur.destroy();
dsTS.disconnect();
dsTS.destroy();
```

This example was taken from the TExecuteTS.java sample.

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
...

dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
while (pCur->isValid()) {
    item = pCur->fetchNext();
    if (item != 0) {
        // Process the DKDDO
        ...
        delete item;
    }
}
delete pCur;
dsTS.disconnect();
```

This example was taken from the TExecuteTS.cpp sample.

Evaluating a text query from a content server

The DKDatastorexx that you use to represent a content server provides an evaluate method to run a query and return a DKResults collection. The following example shows how to evaluate a text query against an earlier DB2 Content Manager content server.

Java

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

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

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";

...
DKAny any = dsTS.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
    element = pIter->next();
    item = (DKDDO*) element->value();
    // Process the DKDDO
    ...
}
delete pIter;
delete pResults;
dsTS.disconnect();
```

Getting match highlighting information

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

When formulating the query string you set MATCH_INFO and MATCH_DICT options. Set MATCH_INFO to YES to return the match highlighting information.

The MATCH_DICT option specifies whether the highlighting information will be obtained using a dictionary. The match information is returned in the DKMATCHESINFO attribute in the DKDDO returned from a text query. The value of the DKMATCHESINFO attribute will be a DKMatchesInfoTS object.

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

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

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect to the content server
//      replace following with your library server,
//      user ID, password
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,
FRNADMIN, PASSWORD)");
// ----- Formulate the query string
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
"OPTION=(SEARCH_INDEX=TMINDEX; MAX_RESULTS=5; +
"MATCH_INFO=YES; MATCH_DICT=NO)";

...

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

Java (continued)

```
// ----- Process the DKDDO
for (i = 1; i <= item.dataCount(); i++)
{
    anyObj = item.getData(i);
    if (anyObj instanceof String)
    {
        ...
    }
    else if (anyObj instanceof Integer)
    {
        ...
    }
    else if (anyObj instanceof Short)
    {
        ...
    }
    else if (anyObj instanceof DKMatchesInfoTS)
    {
        pMInfo = (DKMatchesInfoTS)anyObj;
        // ----- process the Match Hightlighting information
        if (pMInfo != null)
        {
            strDoc = pMInfo.getDocumentName();
            numberSections = pMInfo.numberOfSections();
            // ----- loop thru document sections
            for (j = 1; j <= numberSections; j++)
            {
                pMSect = pMInfo.getSection(j);
                strSection = pMSect.getSectionName();
                numberParagraphs = pMSect.numberOfParagraphs();
                // ----- loop thru section paragraphs
                for (k = 1; k <= numberParagraphs; k++)
                {
                    pMPara = pMSect.getParagraph(k);
                    lCCSID = pMPara.getCCSID();
                    lLang = pMPara.getLanguageId();
                    numberTextItems = pMPara.numberOfTextItems();
                    // ----- loop thru paragraph text items
                    for (m = 1; m <= numberTextItems; m++)
                    {
                        pMText = pMPara.getTextItem(m);
                        strText = pMText.getText();
                        // ----- if match found in text item get offset
                        //          and length of match in text item
                        if (pMText.isMatch() == true)
                        {
                            lOffset = pMText.getOffset();
                            lLen = pMText.getLength();
                        }
                        numberNewLines = pMText.numberOfNewLines();
                    }
                }
            }
        }
    }
}
dsTS.disconnect();
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5;"
      MATCH_INFO=YES;MATCH_DICT=NO)";
...
dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        // Process the DKDDO
        for (i = 1; i <= item->dataCount(); i++)
        {
            anyObj = item->getData(i);
            switch (anyObj.typeCode())
            {
                case DKAny::tc_string :
                {
                    ...
                    break;
                }
                case DKAny::tc_long :
                {
                    ...
                    break;
                }
                case DKAny::tc_short :
                {
                    ...
                    break;
                }
                case DKAny::tc_dobase :
                {
                    ...
                    break;
                }
            }
        }
    }
}
// continued...
```

C++ (continued)

```
// process the Match Highlighting information
pD0Base = a;
pMInfo = (DKMatchesInfoTS*)pD0Base;
if (pMInfo != 0)
{
    strDoc = pMInfo->getDocumentName();
    numberSections = pMInfo->numberOfSections();
    // loop thru document sections
    for (j = 1; j <= numberSections; j++)
    {
        pMSect = pMInfo->getSection(j);
        strSection = pMSect->getSectionName();
        numberParagraphs = pMSect->numberOfParagraphs();
        // loop thru section paragraphs
        for (k = 1; k <= numberParagraphs; k++)
        {
            pMPara = pMSect->getParagraph(k);
            lCCSID = pMPara->getCCSID();
            lLang = pMPara->getLanguageId();
            numberTextItems = pMPara->numberOfTextItems();
            // loop thru paragraph text items
            for (m = 1; m <= numberTextItems; m++)
            {
                pMText = pMPara->getTextItem(m);
                strText = pMText->getText();
                // if match found in text item get offset and
                // length of match in text item
                if (pMText->isMatch() == TRUE)
                {
                    lOffset = pMText->getOffset();
                    lLen = pMText->getLength();
                }
                numberNewLines = pMText->numberOfNewLines();
            }
        }
        break;
    }
    default :
    {
        break;
    }
}
...
delete item;
}
delete pCur;
dsTS.disconnect();
```

Getting match highlighting information for a particular text query result item:

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

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;

dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";

...
pCur = dsTS.execute(cmd);
DKDDO item = null;
Object anyObj = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc;
String strSection;
String strText;
String strDID = "";
String strXNAME = "";
String strDataName = "";
DKPid pid = null;
while (pCur.isValid())
{
    item = pCur.fetchNext();
    if (item != null)
    {
        pid = item.getPid();
        // Process the DKDDO
        for (i = 1; i <= item.dataCount(); i++)
        {
            anyObj = item.getData(i);
            strDataName = item.getDataName(i);
            if (strDID.equals(""))
            {
                strDID = pid.getId();
            }
            if (strXNAME.equals(""))
            {
                strXNAME = p.getObjectType();
            }
            ...
        }
    }
}
// continued...
```

Java (continued)

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

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
...
dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
DKString strDID;
DKString strXNAME;
DKString strDataName;
DKPid pid;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        pid = item->getPid();
        // Process the DKDDO
        for (i = 1; i <= item->dataCount(); i++)
        {
            anyObj = item->getData(i);
            strDataName = item->getDataName(i);
            if (strDataName == "")
            {
                strDID = pid.getId();
            }
            if (strXNAME == "")
            {
                strXNAME = p->getObjectType();
            }
            switch (anyObj.typeCode())
            {
                ...
            }
        }
    }
}
// continued...
```

C++ (continued)

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

Using the result set cursor

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

Important: The information in this section does not apply to DB2 Content Manager 8.3. See Chapter 11, “Working with XML services (Java only)” for details.

Opening and closing the result set cursor to rerun the query

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

Java

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

DKNameValuePair parms[] = null;
...

dkResultSetCursor pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);

pCur.close();
pCur.open();           //re-execute the query
```

C++

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

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

Setting and getting positions in a result set cursor

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

Java

```
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);" +
             "OPTION=(CONTENT=YES);" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC";

DKNVPair parms[] = null;
DKDDO item = null;
int i = 0;
...
// ----- Execute the query; the result cursor is returned
dkResultSetCursor pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
// ----- Use a while loop to iterate thru the collection
while (pCur.isValid())
{
    pCur.setToNext();
    item = pCur.fetchObject();
    if (item != null)
    {
        i = pCur.getPosition();
    }
}
}
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE)";
cmd += "OPTION=(CONTENT=YES)";
cmd += "TYPE_QUERY=DYNAMIC";
cmd += "TYPE_FILTER=FOLDERDOC";
pCur = 0;
DKDDO *item = 0;
long i = 0;
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
    pCur->setToNext();
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

Another way to do this is:

Java

```
Object a = null;
pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
while (pCur.isValid()) {
    pCur.setPosition(DK_CM_NEXT,a);
    item = pCur.fetchObject();
    if (item != null) {
        i = pCur.getPosition();
    }
}
}
```

C++

```
DKAny a;
pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
    pCur->setPosition(DK_CM_NEXT,a);
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

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

Java

```
Object a = null;
pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
a = new Integer(2);
while (pCur.isValid()) {
    pCur.setPosition(DK_CM_RELATIVE,a); // move cursor 2 positions forward
    item = pCur.fetchObject();          // from the current position
    if (item != null) {                  // (relative)
        i = pCur.getPosition();
    }
}
```

C++

```
DKAny a;
long increment = 2;
pCur = dsDL.execute(cmd);
a = increment;
while (pCur.isValid()) {
    pCur->setPosition(DK_CM_RELATIVE,a);
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

Creating a collection from a result set cursor

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

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

Java

```
DKSequentialCollection seqColl = new DKSequentialCollection();
boolean fItems = false;
int how_many = 0;
fItems = pCur.fetchNextN(how_many, seqColl);
```

C++

```
DKSequentialCollection seqColl;
DKBoolean fItems = FALSE;
long how_many = 0;
fItems = pCur->fetchNextN(how_many, seqColl);
```

Querying collections

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

Getting the result of a query

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

Java

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

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

C++

```
// establish a connection
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// create a query object
DKString query1 = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(Title <> NULL));";
DKParametricQuery* pq = (DKParametricQuery*)
    dsDL.createQuery(query1, DK_PARAMETRIC_QL_TYPE, NULL);
pq->execute();
DKAny any = pq->result();
DKResult* rs = (DKResults*) any.value();
```

Evaluating a new query

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

Java

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

C++

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

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

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

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

The following example shows sequential text queries:

Java

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

// ----- The first query
String tquery1 = "SEARCH=(COND=(IBM)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery1, DK_CM_TEXT_QL_TYPE, null);
tq.execute();
DKResults trs = (DKResults) tq.result();
// ----- The second query
String tquery2 = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";
Object obj = trs.evaluate(tquery2, DK_CM_TEXT_QL_TYPE, null);
```

C++

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

DKString tquery1 = "SEARCH=(COND=(IBM)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery* tq =
    (DKTextQuery*) dsTS.createQuery(tquery1,DK_TEXT_QL_TYPE, NULL);
tq->execute();
any = tq->result();
DKResults* trs = (DKResults*) any.value();

DKString tquery2 = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
any = trs->evaluate(tquery2,DK_TEXT_QL_TYPE, NULL);
```

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

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

Using queryable collection instead of combined query

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

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

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

Chapter 4. Working with DB2 Content Manager Version 8.3

This section describes the DB2 Content Manager Version 8 Release 3 connector (ICM connector) application programming interfaces (APIs). The ICM connector is an extension of the Information Integrator for Content framework, so it is essential that you understand the Information Integrator for Content framework concepts described in Chapter 1, “Information Integrator for Content application programming concepts,” on page 1 before continuing with this information.

You can use the ICM connector APIs to build and deploy custom applications that access a DB2 Content Manager content server. You can also use the APIs to integrate your existing applications into a DB2 Content Manager content server.

This section contains the following information:

- Understanding the DB2 Content Manager system
- Understanding DB2 Content Manager concepts
- Planning a DB2 Content Manager application
- Creating a DB2 Content Manager application
- Controlling access to information
- Processing transactions
- Searching for items

Understanding the DB2 Content Manager system

The main components of the DB2 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 DB2 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 manager.

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 DB2 Content Manager system. With the resource manager you can complete the following tasks:

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

The APIs provide applications with access to the DB2 Content Manager system. The APIs are available for Java and C++. Using the APIs, your applications can

take advantage of all of the DB2 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 7 illustrates how the system components fit together. Keep in mind that this is only one implementation of a DB2 Content Manager system. In another system configuration you might have four resource managers for example.

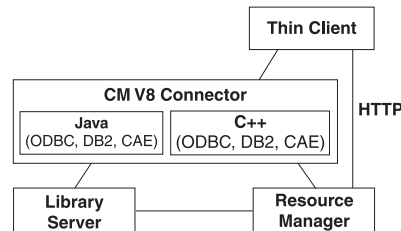


Figure 7. System configuration

Understanding DB2 Content Manager concepts

This section describes important DB2 Content Manager concepts. It is imperative that you understand the DB2 Content Manager concepts before you proceed to the programming tasks. The information described in this section includes:

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

Items

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

When an application creates an item, DB2 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 DB2

Content Manager and used to locate the item within the library server. When writing your application, you use the itemID to access all of the data associated with the item.

Attributes

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

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

You can also define attributes that have multiple values. Such attributes are called multi-valued attributes, which are implemented as child components. For example, you can store multiple addresses, home address, work address, and so forth for a policy owner.

For additional information, see the `SAttributeDefinitionCreationICM` sample.

Item types

An *item type* (index class in earlier DB2 Content Manager versions) is essentially a template for defining and later locating like items. An item type consists of a root component, zero or more child components, and a classification. An item type is the overall structure containing all the components and associated data. For example, in an insurance scenario a policy item type contains items with attributes like policy number, name, claim, and so forth.

In Content Manager, there are four classifications of item types: non-resource, resource, document (also known as document model), and document part. A non-resource item type represents entities that are not stored in a resource manager. A resource item type represents objects stored in a resource manager, like files in a file system, video clips in a video server, LOBs (large objects) in database tables, and so forth. A document item type represents entities that contain document parts that contain resource content just as a single resource item type does. A document part item type represents objects stored in the resource manager, but are parts of a document, contained and owned by a document item type. DB2 Content Manager provides a base set of resource item types: LOB, text, image, stream, and video objects.

For more information about item types, see the `SItemTypeCreationICM` sample.

Root and child components

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

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

A *child component* is an optional second or lower level of a hierarchical item type. Each child component is directly associated with the level above it. Figure 8 on page 110 shows the diagram of the DB2 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.

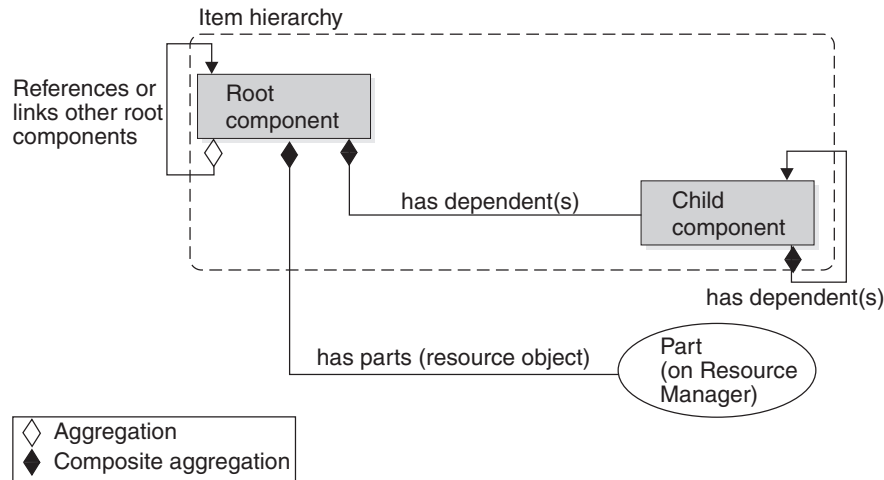


Figure 8. The DB2 Content Manager meta-model: A logical view.

Figure 9 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.

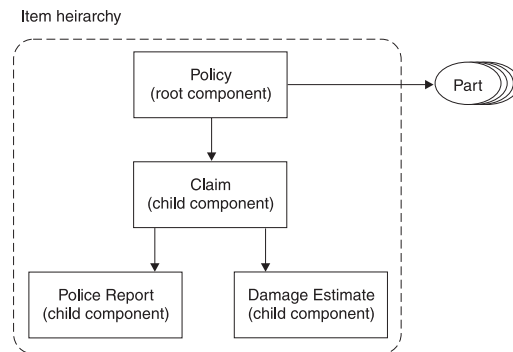


Figure 9. Components hierarchy

For more information about DB2 Content Manager data modeling concepts, the SItemTypeCreationICM sample.

Objects

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

For more information, see the SResourceItemCreationICM sample.

Links and references

You can use a *link* to model one to many associations between items. As shown in Figure 8, links can only be used to associate root components of items. Links do not refer to a particular version of an item. You can define any number of links. Links are a flexible way to link a source to a target. A link simply associates two

items and provides the means to access the items it links or to other items that might link to those two items. Usage of links is determined at runtime by the user application. You can use any number of links in your application.

Links are open and non-restrictive, flexible building blocks that you can use in your applications. As such, you have the option of placing any further restrictions on links within your application.

One of the ways that you can use a link is to represent the foldering or container-containee relationship. If you choose to implement foldering using links, remember that the container does not *own* the containee, which means that the items in the container are not deleted when the container is deleted. For more information about links, see the SLinksICM sample.

A *reference* is between a component (either root or child) and another root component. A reference is represented as a reference attribute in a component defined at design-time. A component definition can have any number, specified during component type definition, of reference attributes that refer to other root components. A reference usually does not indicate ownership, but you can implement an ownership relationship if necessary.

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

References can be defined in both root and child component types referring to another root component type, see Figure 8 on page 110.. References also refer to a specific version of an item, whereas links refer to all versions. For more information about reference attributes, see the SReferenceAttrDefCreationICM sample.

Documents

There are two types of documents that you might have in your system. The first type of document is an item of the semantic type document, which is expected to contain information that forms a document. This type of document can stand alone or contain parts if you have implemented the document model in your data model. For more information on this type of document, see the SItemCreationICM sample in the samples directory.

The other type of document is an item created from a document classified item type (also known as the "document model"). This type of document contains document parts, a specific implementation of the DB2 Content Manager document model. In the document model, items are an extension of non-resource items. Document model parts are resource items. The document parts can include various types of content including text, images, and spreadsheets for example. For more information about the document model, see the SDocModelItemICM sample.

Folders

A *folder* is an item that may contain other items of any type. This may also include other folders. In DB2 Content Manager Version 8, the concept of folders is implemented by using link relationships between items. Items can contain other items to form a containment hierarchy, called a folder hierarchy. For example, a policy item belongs to the policy item type, and potentially has many claims,

making policy a folder that holds other items such as a photo, a social security number, and so forth. Folders are very flexible because any item can be a folder and can contain any number of other items. For more information, see the `SFolderICM` sample.

Versioning

Versioning is the ability to store and maintain multiple versions of an item, including versions of the item's child components. You specify versioning rules when you define an item type. If an item type is enabled for versioning, all items in that item type are versioned.

There are two types of versioning, always or by application. When an item is enabled for versioning always, a new version of the item is created automatically every time the item is updated and stored into the content server. When an item is enabled for versioning by application, the system only creates a new version when specified by the user application.

Versioning is handled by the DB2 Content Manager library server. Each version of an item, whose content is stored in the resource manager, will have its own copy of the content. The following is a list of important versioning characteristics of versioning:

- Versioning involves a root component and its entire hierarchy.
- Item types can have one of three possible versioning policies: version-always, version-never (the default), and application-controlled versioning.
- All the versions of an item in the 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)dco.getPidObject();
String version = pid.getVersionNumber();
....
pid.setVersionNumber(version);
```

- An item type can be configured to keep only a limited number of versions for each item. If an update to an item exceeds the maximum number of allowed, the oldest saved version is dropped and a new version is created by the system.
- If a version-enabled item is reindexed, all previous versions of the item are automatically deleted.
- Child components of an item inherit the version of their parent component.
- The version of a child component type cannot be changed, since it follows the versioning of its parent type.
- Part-level versioning rules can be obtained from the item type relation object that represents the types.

For detailed information about versioning, see the `SItemUpdateICM` and the `SItemTypeCreationICM` sample.

Access control

The DB2 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 DB2 Content Manager user is granted a set of user privileges. These privileges define the operation a user can perform. A user's effective access rights will never exceed the user's defined privileges.

The access control model of DB2 Content Manager is applied to the controlled entity. A *controlled entity* is a unit of protected user data. In DB2 Content Manager, the controlled entity can be at the level of item, item-type, or at the level of the entire library. For example, you can bind an ACL to an item type to enforce access control at the item type level. Operations on controlled entities are regulated by one or more control rules, called access control lists (ACLs). Every controlled entity in Content Manager system must be bound to an ACL.

When a user initiates an operation on an item, the system checks the user's privilege and the ACL bound to the item to determine if the user has the right to do such an operation on the item. Logically, the right to access an item also requires the right to access the item type, where the item is defined. Figure 10 shows an example of how the system determines user's access rights to an item based on privileges and ACLs.

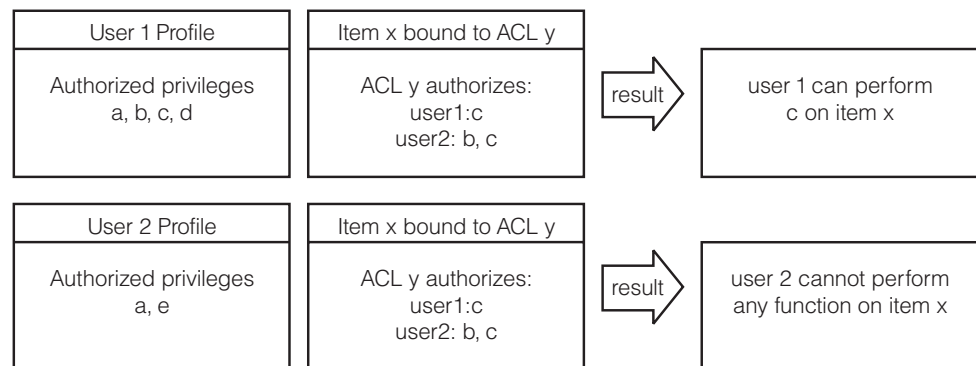


Figure 10. Access control diagram

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 DB2 Content Manager user is granted a set of user privileges. The privileges define the maximum operations a user can perform on information in the DB2 Content Manager system. A user's access rights do not exceed the defined user privileges for the user.

DB2 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 DB2 Content Manager system. You need system administration privileges to complete tasks such as configuring the system, managing the library server configuration, and managing item types. You can use the data access privileges to access and change the system data, like items and item types.

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

The DB2 Content Manager pre-defined privilege sets include: System Admin privilege

AllPrivSet; PrivSetCode: 1

A user with this privilege set can perform all functions on all DB2 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 DB2 Content Manager entities. The privileges contained in this privilege set include:
None.

SystemAdminPrivSet; PrivSetCode: 3

Users with this privilege set can perform all DB2 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 DB2 Content Manager data modeling and item access functions. The privileges contained in this privilege set include:

- System define item type privilege
- Item SQL select privilege
- Item type query privilege
- Item query privilege
- Item add privilege
- Item set user defined attr privilege
- Item set system defined attr privilege
- Item delete privilege
- Item move privilege
- Item link to privilege
- Item linked privilege
- Item own privilege
- Item owned privilege

- Item add link privilege
- Item change link privilege
- Item remove link privilege
- Item check out privilege

Table 6. Privilege Codes

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

Users and user groups

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

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

Access control lists (ACLs)

When a user creates an item in the DB2 Content Manager system, that user must define the access that other users will have to that item, and what operations they can perform on that item. The list of users that have access to the item and the operations that they can perform on the item is called an *access control list* (ACL). An ACL can contain one or more individual user IDs or user groups and their associated privileges. You can associate items, item types, and worklists with an ACL. Privilege sets define an individual's maximum ability to use the system, an ACL restricts that individual's access to an item. For example, if its ACL allows the photograph item to be deleted but John doesn't have the delete privilege in his privilege set, then John cannot delete the photograph.

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

The users specified in access control rules can be individual users, user groups, or public. The interpretation is determined by the UserKind field of a rule. The types of rules, for illustration purposes, can be given the names ACL Rule for User, ACL Rule for Group, and ACL Rule for Public respectively. By specifying public, the ACL Rule for Public authorizes all the users to perform operations specified in the

ACL Privileges on the bound entity, provided the users pass their User Privileges check. The ACL privileges on the bound entity to Public can be configured in the System level. The capability of opening a bound entity to Public can be configured system-wide. The configuration parameter is named `PubAccessEnabled` (defined in table `ICMSTSysControl`). When disabled, all the ACL Rules for Public are ignored during the access control process.

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

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

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

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

SuperUserACL

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

NoAccessACL

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

PublicReadACL

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

Planning a DB2 Content Manager application

This section helps you identify requirements for creating a DB2 Content Manager application and provides information about how DB2 Content Manager operates. A key part of planning your application is creating a data model that meets the needs of your business. For more information about the DB2 Content Manager data model, see *Planning and Installing Your Content Management System* and the `tSItemTypeCreationICM` sample in the samples directory.

The following topics are covered in this section:

- Determining the features of your application.
- Handling errors.

Determining the features of your application

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

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

- What types of documents does your organization use?
- What type of content is in your existing documents?
- How do you process documents?
- Can you automate your document process?
- How do you receive, display, store, and distribute documents?
- How often do you retrieve documents after they are stored?
- What is the volume of documents that your organization manages?
- What types of storage media do you want to use to store your large objects?
- Are there other applications your organization uses?
- How many users and what type of access control do you plan to have?

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

Handling errors

When handling errors, the most important exception to catch is the `DKException` class. Do not use exceptions for program logic, and do not rely on catching exceptions to detect if something exists in the content server or for any reason other than for truly exceptional cases. Using exceptions in program logic decreases performance and can render tracing and log information useless for debugging and support.

Carefully review all of the exception information. There are numerous sub-classes of `DKException` and depending on the program, it might be best to handle each exception individually. Table 7 contains `DKException` information.

Table 7. *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 7. *DKException* information (continued)

DKException	Description
Error State	Might contain additional error information about the state of the OO API or library server error. If the library server detects an error, the following four pieces of information are packaged here: Return code Reason code Ext / SQL return code Ext / SQL reason code
Error Code	Might contain the library server return code.
Stack Trace	Important information indicating the failure point in the user program and exactly where the error was last detected or handled by the OOAPI.

When working in Java, you must also handle the `java.lang.Exception`. The `SConnectDisconnectICM` sample in the `samples` directory demonstrates how to catch and print errors. For information about logging and tracing, see *Messages and Codes*.

Working with the DB2 Content Manager samples

Content Manager provides a comprehensive set of code samples to help you complete key DB2 Content Manager tasks. The samples are a great source of ICM API education because they provide reference information, programming guidance, API usage examples, and tools.

You can view the samples in the *Application Programming Reference*, in the product Information Center. Additionally, the samples are located in the `IBMCMROOT/samples/cpp/icm` and `IBMCMROOT/samples/java/icm` directories. Note, however, that you must have selected the Samples and Tools component during Information Integrator for Content installation in order to have the samples in the directory.

Important: The DB2 Content Manager Express samples are located in the following directory: `\samples\java\cmx` and `\samples\cpp\cmx`

To get the most out of the samples, be sure to read the Samples Readme. It contains a complete reference index to help you quickly find the sample that contains the concept, or topic, that you are looking for. Every sample is thoroughly documented and provides in-depth conceptual information and an explanation of each task step. Additional information contained in each sample includes:

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

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

- Data modeling.
- Connecting to a server and handling errors.

- Defining attributes and attribute groups.
- Working with reference attributes.
- Defining your data model.
- Working with items.
- Working with resource items.
- Working with folders.
- Working with links.
- Defining the .
- Defining an SMS collection.
- Searching for items.

The insurance scenario sample

DB2 Content Manager provides code samples for one possible "real world" implementation using an insurance company. The information used to create the insurance company sample is fabricated and created only to help explain key Content Manager features. For a complete list of the samples that make up the insurance scenario, see the Samples Readme.

Creating a DB2 Content Manager application

The APIs that implement DB2 Content Manager Version 8 Release 3 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 DB2 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 DB2 Content Manager hierarchical data model. For example, an insurance company's data model includes *policies*, which in the DB2 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 DB2 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 DB2 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 1. The information in this section is specific to DB2 Content Manager Version 8 Release 3.

A DDO is essentially a container of attributes. An attribute has a name, value, and several properties. One of the most important properties of attributes is the attribute type. A DDO has a persistent identifier (PID) to indicate the location where the object resides in persistent storage. A DDO has some methods to populate itself, and corresponding methods to retrieve an item’s information. The DDO methods include add, retrieve, update, and delete. You use these methods to move an item’s data in and out of persistent storage, like DB2 Content Manager.

In memory, DB2 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 DB2 Content Manager system

One of the first things that you need to do when you build a DB2 Content Manager application is connect to the server. This section helps you with the various tasks involved in connecting to, and disconnecting from, a DB2 Content Manager server.

To access the DB2 Content Manager server, your application needs to create a content server, which acts as a common server. To create and connect to a content server:

1. Create a content server object.

Java

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

C++

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

2. Set up the connection parameters.

Java

```
String database = "icm1sdb";
String userName = "icmadmin";
String password = "password";
```

C++

```
char * database = "icm1sdb";
char * userName = "icmadmin";
char * password = "password";
```

3. Call the connect operation on the content server. `databaseNameStr` is the name of the database you want to connect to.

Java

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

C++

```
dsICM->connect(database, userName, password, "");
```

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

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

To disconnect from the system, call the `disconnect` operation in the content server.

For more information, see `SConnectDisconnectICM`, the complete sample from which the above code snippets were extract.

Changing a password

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

Java

```
changePassword(String userID, String oldPwd, String newPwd)
```

C++

```
changePassword(const char* userID, const char* oldPwd, const char* newPwd)
```

Working with items

This section describes how to create, update, and delete items.

For additional information about working with items, see the `SItemCreationICM` sample.

Creating an item type

Before you can create any items, you must have already created item types, which are much like categories, to place the items under. For example, a `claim` item can be placed under the item type policy. You can think of this relationship as a parent-child relationship where the child is the item, or the `claim`, and the parent is the item type, or the policy.

When you create an item type, you can define a classification for the item type. An *item type classification* is a categorization within an item type that further identifies the items of that item type. All items of the same item type have the same item type classification. Content Manager supplies the following item type classifications: `item`, `resource item`, `docmodel`, and `docpart`. If you do not specify an item type classification when you create an item type, the item type classification defaults to `item` (DDO). The pre-defined item type classifications and the corresponding ID constant are listed in Table 8.

Table 8. Item type classifications

Classification	ID Constant	Number	Description
Item	DK_ICM_ITEMTY PE_CLASS_ITEM	0	A standard item (DDO).
Resource	DK_ICM_ITEMTY PE_CLASS_RES OURCE_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_ITEMT YPE_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 <code>DKConstant.DK_CM_DKPARTS</code> .
Part	DK_ICM_ITEMTY PE_CLASS_DOC_ PART	3	Items (parts) in the document model classification.

Table 8. Item type classifications (continued)

Classification	ID Constant	Number	Description
Note: Constants are located in com\ibm\mm\sd\common\DKConstantICM.java for Java and dk\icm\DKConstantICM.h for C++.			

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

1. Create an item type and pass it a reference to the content server.
2. Give the item type a name. Item type names should be less than 15 characters in length. The item type name is used to create DDOs of the item type.
3. Add attributes to the item type, and set any desired qualifiers, like nullable, textsearchable, and unique.
4. Add the item type to the persistent content server.

Java

```
//This example creates an item type definition and names it.
//The item type name must be less than 15 characters in length.
DKItemTypeDefICM bookItemType = new DKItemTypeDefICM(dsICM);
    bookItemType.setName("book");
    bookItemType.setDescription("This is an example item type name.");
//Below, a text-searchable attribute called book title is added. The
//attribute is defined to require a unique name and also a value. The value
//does not have to be unique.
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_title");
    attr.setTextSearchable(true);
attr.setUnique(true);
attr.setNullable(false);
bookItemType.add(attr);
//Here, a book_num_pages attribute is added to the book item
//type with the following characteristics: text searchable, unique,
//and a value is not required.
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_num_pages");
attr.setTextSearchable(false);
attr.setUnique(false);
attr.setNullable(false);
    bookItemType.addAttr(attr);
    bookItemType.add();
```


C++

```
DKDatastoreICM* pDs;  
DKDatastoreICM* pDs;  
...  
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*)pDs->datastoreDef();  
//create new ItemType  
DKItemTypeDefICM * bookItemType = new DKItemTypeDefICM(pDs);  
bookItemType->setName("book");  
bookItemType->setDescription("This is an example item type name.");  
//Create new Attribute; add it to datastore and to the ItemType  
DKAttrDefICM * attr = (DKAttrDefICM *)dsDefICM->createAttr();  
    attr->setName("book_title");  
    attr->setType(DK_CM_VARCHAR);  
    attr->setSize(80);  
    attr->setTextSearchable(TRUE);  
    attr->setUnique(TRUE);  
    attr->setNullable(FALSE);  
//Persist the attribute to the datastore  
attr->add();  
//Add the newly created attribute to the item type.  
bookItemType->addAttr(attr);  
//Create new Attribute; add it to datastore and to ItemType  
    attr = (DKAttrDefICM *)dsDefICM->createAttr();  
    attr->setName("book_num_pages");  
    attr->setType(DK_CM_INTEGER);  
    attr->setTextSearchable(FALSE);  
    attr->setUnique(FALSE);  
    attr->setNullable(FALSE);  
    attr->add();  
bookItemType->addAttr(attr);  
//Add the entity, bookItemType, to the datastore.  
bookItemType->add();
```

See the `SitemTypeCreationICM` sample for more information.

Listing item types

To get a list of available, defined item types:

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

Java

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

C++ Example 1

```
long larraySize = 0;
DKString * itemTypeNames = dsICM->listEntityNames(larraySize);
for (int i = 0; i < larraySize; i++) {
    cout <<(char*)itemTypeNames[i] <<endl;
}
delete [] itemTypeNames;
```

C++ Example 2

```
DKSequentialCollection * itemTypeColl =
    (DKSequentialCollection *)
        dsICM->listEntities();
dkIterator * iter = itemTypeColl->createIterator();
while (iter->more()) {
    dkEntityDef* itemType=(dkEntityDef*)((void*)(*iter->next()));
    cout <<(char*)itemType->getName() < delete(itemType);
}
delete(iter);
delete(itemTypeColl);
```

For more information, see the SItemTypeRetrievalICM sample.

Creating attributes

Attributes in DB2 Content Manager are created as independent objects and have generic properties. When creating an item type, you determine which attributes to include into the item type. Once an attribute is included in an item type, you can further define the attribute properties for that particular item type. For example, you can set the attribute ISBN under the item type Journal to nullable, but under the item type Book, you can set it to non-nullable. Other properties that you can specify for an item type include readable, writable, text-searchable, uniqueness, representative property, and so forth. See the Application Programming Reference for the complete list of properties that you can specify in an attribute.

To create an attribute, complete the following steps:

1. Create an attribute definition object using the DKAttrDefICM class.
2. Describe the object that you create by setting its name, description, type, size, and so forth. Attribute names are limited to 15 characters. If you need to include more information, use the description field. The following table contains examples of types of attributes that you can create:

Table 9. Attribute types

Type	Constant	Object
BLOB	DKConstant.DK_CM_BLOB	byte bytes[]
Char	DKConstant.DK_CM_CHAR	java.lang.String
CLOB	DKConstant.DK_CM_CLOB	java.lang.String
Date	DKConstant.DK_CM_DATE	java.sql.Date
Decimal	DKConstant.DK_CM_DECIMAL	java.math.BigDecimal
Double	DKConstant.DK_CM_DOUBLE	java.lang.Double
Integer	DKConstant.DK_CM_INTEGER	java.lang.Integer
Short	DKConstant.DK_CM_SHORT	java.lang.Short

Table 9. Attribute types (continued)

Type	Constant	Object
Time	DKConstant.DK_CM_TIME	java.sql.Time
Timestamp	DKConstant.DK_CM_TIMESTAMP	java.sql.Timestamp
Varchar	DKConstant.DK_CM_VARCHAR	java.lang.String

3. Add the new attribute definition to the persistent content server.

Java

```
//This example defines an attribute for the title of a book.
attr = new DKAttrDefICM(dsICM);
attr.setName("book_title");
attr.setDescription("The title of the book.");
attr.setType(DKConstant.DK_CM_VARCHAR);
attr.setSize(100);
attr.add();
//This example defines an attribute for the number of pages in a book.
attr = new DKAttrDefICM(dsICM);
    attr.setName("book_num_pages");
    attr.setDescription("The number of pages in the book.");
attr.setType(DKConstant.DK_CM_INTEGER);
    attr.setMin((short) 0);
    attr.setMax((short) 100000);
attr.add();
```

C++

```
//This example defines an attribute for the title of a book.
DKDatastoreICM * dsICM; .....
DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
    attr->setName("book_title");
    attr->setDescription("The title of the book.");
    attr->setType(DK_CM_VARCHAR);
    attr->setSize((long) 100);
    attr->add();
//This example defines an attribute for the number of pages in a book.
DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
    attr->setName("book_num_pages");
    attr->setDescription("The number of pages in the book.");
    attr->setType(DK_CM_INTEGER);
    attr->setMin((long) 0);
    attr->setMax((long) 100000);
    attr->add();
```

For more information about creating attributes, see the `SAttributeDefinitionCreationICM` sample.

Creating, updating, and deleting attribute groups

Attribute groups make it easier for you to add entire groups of attributes to items and sub-components. An attribute can belong to any number of attribute groups, from zero to any number. An attribute can be added to multiple attribute groups. A data item (DDO) can contain multiple attribute groups. The same attribute name can appear in the DDO, but each attribute is completely separate, based on the namespace. When an attribute is added to an attribute group, it impacts only the

component types that are created after the addition. Pre-existing component types remain unchanged. The following example demonstrates how to create an attribute group:

Java

```
// Create a datastore definition object given the connected datastore
DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
//Creating a new attribute group
DKAttrGroupDefICM attrGroup = new DKAttrGroupDefICM(dsICM);
// Set a name, maximum 15 characters
attrGroup.setName("Book_Details");
//Set a description for the new attribute group
attrGroup.setDescription("Detailed book information");
// Retrieve the definition of an attribute that will be
//added to this attribute group.
DKAttrDefICM title = (DKAttrDefICM) dsDefICM.retrieveAttr("book_title");
// Retrieve the definition of another attribute that will be added
//to this attribute group.
DKAttrDefICM publisher = (DKAttrDefICM) dsDefICM.retrieveAttr("publisher");
// Add the Attribute Definitions to the Attribute Group
attrGroup.addAttr(title);
attrGroup.addAttr(publisher);
// add it to the persistent datastore
attrGroup.add();
```

C++

```
// Create a datastore definition object given the connected datastore
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
//Creating a new attribute group
DKAttrGroupDefICM* attrGroup = new DKAttrGroupDefICM(dsICM);
// Set a name, maximum 15 characters
attrGroup->setName("Book_Details");
//Set a description for the new attribute group
attrGroup->setDescription("Detailed book information");
// Retrieve the definition of an attribute that will be
//added to this attribute group.
DKAttrDefICM* title =
    (DKAttrDefICM *) dsDefICM->retrieveAttr("book_title");
// Retrieve the definition of another attribute that will be added
//to this attribute group.
DKAttrDefICM* publisher =
    (DKAttrDefICM *) dsDefICM->retrieveAttr("publisher");
// Add the Attribute Definitions to the Attribute Group
attrGroup->addAttr(title);
attrGroup->addAttr(publisher);
// add it to the persistent datastore
attrGroup->add();

delete(attrGroup);
```

To update the name and description of an attribute group, call the `update()` method in `DKAttrGroupDefICM` and provide an array of new attribute IDs. If this attribute group has already been associated with a component type, then you can't update this attribute group.

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

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

For more information, see the `SAttributeGroupDefCreationICM` sample.

Listing the attributes in a content server

The following example demonstrates how to get a list of attributes in a content server.

Java

```
DKDatastoreICM dsICM;
DKDatastoreDefICM dsDefICM=DKDatastoreDefICM(dsICM.datastoreDef());
//Get a collection containing all Attribute Definitions.
DKSequentialCollection attrDefColl =
    (DKSequentialCollection)dsDefICM.listAttrs();
if ((attrDefColl!=null) &&(attrDefColl.cardinality()>0)){
    //Create an iterator to iterate through the collection
    dkIterator iter = attrDefColl.createIterator();
    while(iter.more()){
        //while there are still items in the list, continue
        dkAttrDef attrDef = (dkAttrDef) iter.next();
        System.out.println("-"+attrDef.getName()+" "+attrDef.getDescription());
    }
}
```

C++

```
DKDatastoreICM * dsICM; .....
DKDatastoreDefICM*dsDefICM = (DKDatastoreDefICM *)dsICM->datastoreDef();
//Get a collection containing all Attribute Definitions.
DKSequentialCollection*attrDefColl =
    (DKSequentialCollection *)dsDefICM->listAttrs();
if (attrDefColl &&(attrDefColl->cardinality()>0)){
    //Create an iterator to iterate through the collection
    dkIterator*iter =attrDefColl->createIterator();
    while(iter->more()){
        //while there are still items in the list, continue
        dkAttrDef* attrDef = (dkAttrDef *)iter->next()->value();
        cout <<" "<<attrDef->getName()<<": "<<attrDef->getDescription()<<endl;
        delete(attrDef);
    }
    delete(iter);
    delete(attrDefColl);
}
delete(dsDefICM);
```

Listing attribute names for an item type

The following example demonstrates how to get a list of attribute names for an item type. For more information, see the `SItemTypeRetrievalICM` ICM API education sample.

Java

```
//Get a collection containing all attr. defs for the Item Type.
DKSequentialCollection attrColl = (DKSequentialCollection)
    dsICM.listEntityAttrs(itemTypeName);
//Accessing attribute each and printing the name and description.
System.out.println("\nAttributes of Item Type '"+itemTypeName+":'
    (" +attrColl.cardinality()+')');
//Create an iterator to iterate through the collection
dkIterator iter = attrColl.createIterator();
while(iter.more()) {
    //while there are still items in the list, continue
    DKAttrDefICM attr = (DKAttrDefICM)iter.next();
    System.out.println("-"+attr.getName()+":"+attr.getDescription());
}
```

C++

```
//Get a collection containing all Attribute Definitions for the Item Type.
DKSequentialCollection*attrColl =(DKSequentialCollection*)
    dsICM->listEntityAttrs(itemTypeName);
//Accessing attribute each and printing the name and description.
cout <<"\nAttributes of Item Type '"<<itemTypeName <<":'
    ("<<attrColl->cardinality()<<')<<
//Create an iterator to iterate through the collection
dkIterator* iter =attrColl->createIterator();
while(iter->more()) {
    //while there are still items in the list, continue
    DKAttrDefICM* attr =(DKAttrDefICM*)iter->next()->value();
    cout <<"-"<<attr->getName()<<": "<<attr->getDescription()<<endl;
    delete(attr);
}
delete(iter);
delete(attrColl);
```

Creating an item

An item is the entire tree of component DDOs, with at least one root component. That root component DDO must be assigned an Item Property Type or Semantic Type. When you create an item, you must assign it an item type property. The valid itemtype properties are document, folder, or item. You must specify the item type property as the second parameter of the content server's `createDDO` function. The value for the property type is stored in the DDO's property named, `DK_CM_PROPERTY_ITEM_TYPE`. Do not confuse this item type property with the overall item type definition that describes the structure of the item. Table 10 shows a list of available item property types.

Table 10. Item type property definitions

Property type	Constant	Definition
Document	DK_CM_DOCUMENT	Item represents a document or stored data. The information contained in this item might form a document. This item can be considered a common document since it does not rigidly mean an implementation of a specific document model.

Table 10. Item type property definitions (continued)

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

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

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

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

Java Example 1

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

Java Example 2

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

C++ Example 1

```
DKDDO* myDocumentDDO = dsICM->createDDO("EmployeeDoc",DK_CM_DOCUMENT);
```

C++ Example 2

```
DKDDO* myFolderDDO = dsICM->createDDO("DeptFolder",DK_CM_FOLDER);
```

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

Java

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

C++

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

4. Add the child component to the parent.

Java

```
short dataid = myDocumentDDO.dataId(DK_CM_NAMESPACE_CHILD,"Dependent");  
DKChildCollection children =  
    (DKChildCollection) myDocumentDDO.getData(dataid);  
children.addElement(myDDO);
```

C++

```
short dataid = myDocumentDDO->dataId(DK_CM_NAMESPACE_CHILD,"Dependent");  
DKChildCollection* children =  
    (DKChildCollection*)(dkCollection*)  
myDocumentDDO->getData(dataid);  
children->addElement(myDDO);
```

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

Java

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

C++

```
myDDO->setData(.....);
```

6. Save the item into the persistent store.

Java

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

C++

```
myDocumentDDO->add();
```

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

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

You can specify the semantic type as the second parameter of the content server's createDDO function. The semantic type value is stored in the DDO's property DKConstantICM.DK_ICM_PROPERTY_SEMANTIC_TYPE, which can contain the same value as the item property type. The ICM connector supports folder and document semantic types. You can also create your own semantic types. Table 11 lists the available semantic types.

Table 11. Pre-defined semantic types

Semantic type	Constant	Definition
Document	DK_ICM_SEMANTIC_TYPE_DOCUMENT	Indicates that this item is a document
Folder	DK_ICM_SEMANTIC_TYPE_FOLDER	Indicates that this item is a folder
Annotation	DK_ICM_SEMANTIC_TYPE_ANNOTATION	Indicates that this item or part is an annotation to the base part
Container	DK_ICM_SEMANTIC_TYPE_CONTAINER	Indicates that this item is a container, which can contains other items. The container-containee relationship is represented using links.
History	DK_ICM_SEMANTIC_TYPE_HISTORY	Indicates that this item or part is a history of the base part.
Note	DK_ICM_SEMANTIC_TYPE_NOTE	Indicates that this item or part is a note to the base part.
Base	DK_ICM_SEMANTIC_TYPE_BASE	Indicates that this item or part is the base part that may have an annotation, note, or history associated with it.
User defined	User defined	A user defined semantic type interpreted by the application.

Notes:

- In Java, the constants are defined in DKConstantICM.java.
- In C++, the constants are defined in DKConstantICM.h.
- In Content Manager Version 7, semantic type is called affiliated type.

If you create an item in an item type that has been defined as a resource item type, the correct XDO is returned. For more information about resources, see the SResourceItemCreationICM sample. You might also find it useful to review the information about item creation in the SItemCreationICM sample.

Setting and retrieving item attribute values

The following example demonstrates how to set and retrieve item attribute values.

Java

Attribute values are stored and retrieved as java.lang.Objects. To set or retrieve attribute values, use the DDO's setData and the getData() methods respectively. Set the value using an object with the type corresponds to the attribute type.

Example:

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
    nameOfAttrStr),valueObj);
Object obj = ddo.getData(ddo.dataId(DK_CM_NAMESPACE_ATTR,nameOfAttrStr));
```

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

C++

When setting values for individual attributes, use the individual attribute definition name. In order to access attributes that belong to an attribute group, use this format: <Attribute Group Name>.<Attribute Name>.

Example:

```
//The code snippet below shows the value of the character attribute
//"book_title" for the item represented by the DDO ddoDocument is set to
//the value "Effective C++"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("book_title")),DKString("Effective C++"));
//The code snippet below shows the value of the integer attribute
//"book_num_pages" for the item represented
//by the DDO ddoDocument is set to the value "250"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("book_num_pages")), (long)250);
//This code snippet shows how the value of the "book_title" attribute of the
//item represented by the DDO ddoDocument is retrieved into the "title"
//string variable.
DKString title = (DKString) ddoDocument->getData(ddoDocument->
    dataId(DK_CM_NAMESPACE_ATTR,DKString("book_title")));
```

DB2 Content Manager Version 8 Release 3

Setting and retrieving foreign key attribute values

In DB2 Content Manager Version 8 Release 3, you can pre-define *foreign key attribute values* to populate client drop-down menus with specific choices. These values would be associated with an attribute of an item type. For example, an auto claims item type could have a city attribute with the drop-down values of New York and Chicago. Similarly, a life claim item type could have a city attribute with drop-down values of Boston, Atlanta, and Dallas.

DB2 Content Manager Version 8 Release 3 provides methods in the DKForeignKeyDeflCM class that can retrieve foreign key attribute values for an item. Starting with Version 8.3, the system administration client and library server can support multiple attribute definitions in a foreign key constraint.

Additionally, you can cascade relationships between the drop-down lists. For example, if you select California for the State attribute value, then this could

populate the City drop-down with San Jose and Los Angeles. To accomplish this, you could define a foreign key item type called StateCity with two attributes: State and City. You could then define items for all valid combinations of state and city. Other item types that have State and City attributes can use the StateCity item type as a foreign key, and restrict the values allowed for city and state. You can return the foreign key fields will be returned in the order of precedence (State then City).

Foreign keys can consist of one or more attribute relationships, which use the following concepts:

Column sequence number

A unique identifier that you assign to describe the source and target attribute relationship in a foreign key.

Source attribute name

Reference to an attribute in a source item or table. For example, State.

Target attribute name

Reference to an attribute in a target item or table. For example, City.

Display flag

Boolean value that tags the foreign key attributes to populate into drop-down menus. If you set this to false, then a text entry field displays instead.

The following examples use `getSpecificAttrsForForeignKey()` to return an a set of results based on a search of provided attribute (column) names in foreign key and their corresponding values. This method can only be used in conjunction with foreign keys related to DB2 Content Manager defined tables.

Java

```
ArrayList getSpecificAttrsForForeignKey(DKNVPair[] fkeyValues,  
String[] columnNames ) throws Exception, DKException
```

Returns an ArrayList of String arrays.

C++

```
dkCollection * getSpecificAttrsForForeignKey(DKNVPair[] fkeyValues,  
long fkeyValuesLen, char * columnNames[], long numOfColumns)
```

Returns a `dkCollection` of `SpecificAttrResults` objects, where each object would contain a `char* result[]` containing the results for each column and the length of this results array. You must can then retrieve the results using `getResultArray()` and `getNumOfResults()` by creating an instance of the `SpecificAttrResults` class:

```
DKExport void SpecificAttrResults (long no_of_results) //constructor  
DKExport void addResult( char * res) //adds a string to the results[]  
char *[] getResultArray() //retrieves the results[] for traversing  
long getNumOfResults() //retrieves the number of length of no_of_array  
~ SpecificAttrResults() //destructor
```

The `DKNVPair` parameter represents an array of `DKNVPairs` containing the names of attributes (columns) in the foreign keys and their corresponding values that will be used to formulate a query. You can optionally specify attributes in the `columnNames` parameter to restrict which multi-column'ed values to return. The

default is to return all results. For an example of results, see Table 12.

Table 12. Example results for parameters passed into `getSpecificAttrsForForeignKey()`

DKNVPairs	Attribute names	Result
DKNVPair fKeyValues[0]("Country", "US");	columnNames[0] = "city"	Returns the names of all cities in the U.S.
DKNVPair fKeyValues[0]("Country", "US") DKNVPair fKeyValues[1]("State", "California")	columnNames[0] = "City" columnNames[1] = "Zip"	Returns the names of all cities (and their zip codes) in the state of California only
DKNVPair fKeyValues[0]("Country", "US") DKNVPair fKeyValues[1]("State", "California") DKNVPair fKeyValues[2]("City", "San Jose")	columnNames = null;	Given that the foreign key contains the attributes: Country, State, City, addresses, zip, and phone number, then returns all values for address, zip, and phone number.

The following examples return foreign key string arrays in the following order: column sequence number, source attribute, target attribute, and display flag value.

Java

```
public Vector listForeignKeyAttrDetails( ) throws Exception, DKException
```

C++

```
dkCollection* listForeignKeyAttrDetails()
```

The following examples define new associations between attributes of two component types into a foreign key definition.

Java

```
public void addSrcAndTgtAttrName(String srcAttrName, String tgtAttrName,
int columnSeq, boolean displayFlag) throws Exception, DKException
```

C++

```
void addSrcAndTgtAttrName(char * srcAttrName, char* tgtAttrName, long
columnSeq, DKBoolean displayFlag)
```

The following examples update the value of the display flag in the in-memory foreign key definition for the attribute relation designated by the column sequence number. The foreign key definition still needs to be updated in persistent storage to make the new value permanent. This method can only be used in conjunction with foreign keys related to DB2 Content Manager defined tables.

Java

```
public void setDisplayFlag(int columnSeq, boolean displayFlag)
throws Exception, DKException
```

C++

```
void setDisplayFlag(long columnSeq, DKBoolean displayFlag)
```

The following examples return the boolean value for the display flag (from the foreign key definition) for the attribute relation denoted by the column sequence number.

Java

```
public boolean getDisplayFlag(int columnSeq) throws Exception, DKException
```

C++

```
DKBoolean getDisplayFlag(long columnSeq)
```

In addition, you can use the following CMBAttribute methods to determine predefined values, which are values assigned to the source attribute that can be referenced later. For example, the City attribute can have predefined values such as Los Angeles and New York that could display in a drop-down menu.

hasPredefinedValues()

Boolean value that indicates whether an attribute has pre-defined values.

getPredefinedValues()

Returns predefined values for an attribute.

getDependentValues()

Returns a list of CMBAttributes whose predefined values depend on this attribute.

getControllingAttributes()

Returns a list of CMBAttributes that the predefined values of this attribute depend on.

For more information on these methods, see the TGetPreDefValues.java and TGetPreDefValues.cpp samples.

Modifying an item's attributes

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

Java

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

C++

```
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,"book_title"),
    DKString("More Effective C++"));
```

Updating an item

To update an item:

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

Java

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

C++

```
ddo->update();
```

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

Java

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

C++

```
ddo->update(DK_CM_VERSION_NEW);
```

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

Java

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

C++

```
ddo->update(DK_CM_VERSION_LATEST);
```

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

Java

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

C++

```
int options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;  
ds->updateObject(ddo, options);
```

Important: You must check out the item before calling the update method and check the item back in when you are done updating it. See “Checking in and checking out items” on page 145. For more information on updating items, see the `SItemUpdateICM` ICM API education sample.

Defining a resource item type

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

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

1. Get the content server definition object from the connected content server.

Java

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

C++

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
```

2. Create a new item type definition.

Java

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

C++

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

3. Add an attribute.

Java

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

C++

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

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

Java

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

C++

```
itemType->setXDOClassName(DK_ICM_XDO_LOB_CLASS_NAME);
itemType->setXDOClassID(DK_ICM_XDO_LOB_CLASS_ID);
itemType->add();
```

Creating a resource item

Creating resource items is very similar to creating regular items. XDOs extend DDOs, and depending on the type of resource item, the XDO can be extended further. Table 13 contains the resource item types and the class hierarchy used to create them. For more information, see the `SResourceItemCreationICM` sample (in the samples directory).

Table 13. Resource item type class hierarchy

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

The steps below take you through the process of creating resource item. There are multiple ways to set and store resource content. The following process is an example for storing directly from file at the time the item is added to the content server.

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

Java

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

C++

```
DKLobICM* lob = (DKLobICM*)  
dsICM->createDDO("SampleResource",DK_CM_DOCUMENT);
```

2. Set the content or data into the object. Once the lob is created, you can set the MIME type for the resource. In this case, the resource is a MS Word document. The MIME type describes the type of content that is being stored.

Java

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

C++

```
lob->setContentFromClientFile(fileName);  
lob->setMimeType("application/msword");
```

For additional information about MIME types, see the SResourceItemMimeTypesICM sample.

3. Add the data to the content server. In this case, a sample Word document. Note that the resource item content is stored in .

Java

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

C++

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

Important: In C++, you must clean up the memory.

```
delete(lob);
```

For more information, see the SItemUpdateICM sample.

Searching for items

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

1. Connect to a DKDatastoreICM object.

2. Process the correct query. Make sure that your query conforms to the query language. For more information see the Searching for data section.
3. Save the query results in a DKResult object.

Java

```
DKNVPair options[] = new DKNVPair[3];
options[0] =
    new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "0"); // No Max (Default)
options[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,
    new Integer(DKConstant.DK_CM_CONTENT_ATTRONLY));
options[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
DKResults results = (DKResults)dsICM.evaluate(query,
    DKConstant.DK_CM_XQPE_QL_TYPE, options);
```

C++

```
DKNVPair *options    = new DKNVPair[3];
// only one result will be returned
options[0].set(DK_CM_PARM_MAX_RESULTS, "1");// Retrieve only one item
options[1].set(DK_CM_PARM_RETRIEVE , (long)DK_CM_CONTENT_ATTRONLY);
// Last option has to be this value
options[2].set(DK_CM_PARM_END , NULL);
//Note if a query is expected to return more than one result,
//the DKDatastoreICM::execute method
//should be used. The execute method returns a dkResultSetCursor.
DKResults* results = (DKResults*)(dkCollection*)dsICM->evaluate
    (query, DK_CM_XQPE_QL_TYPE, options);
```

For additional information, see the SSearchICM sample.

Retrieving an item

To ly retrieve an item or refresh an item retrieved through a query, you must have access to the PID object or PID string of the item to be retrieved or refreshed. If only know the ItemID, you can perform a query to retrieve the complete PID information. Given the item type name and item ID, the following scenario shows how to retrieve the DDO if you want to retrieve ly.

Java

1. After connecting to a `datastoreICM` object, search for the item using the appropriate item ID. For information about writing queries, see “Querying the DB2 Content Manager server” on page 187.
2. Save the query results in a `DKResults` object. In the code example below, `queryStr` is the search string in the correct query language format.

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

```
ddo.retrieve();
```
4. Suppose you have a PID string that you obtained from a DDO as shown in this example:

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

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

1. Create a DDO using the `DKDatastoreICM`’s `createDDO` method. Do not use the `DKDDO` constructor. In the example below, the object `dsICM` is already connected to a DB2 Content Manager datastore. Also, the PID is a string named `pidStr`.

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

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

C++

1. After connecting to a `datastoreICM` object, search for the item using the appropriate item ID. For information about writing queries, see “Querying the DB2 Content Manager server” on page 187.
2. Save the query results in a `DKResult` object. In the code example below, `queryStr` is the search string in the correct query language format.

```
DKResults* results = (DKResults*)(dkCollection*)  
dsICM->evaluate(queryStr, DK_CM_XQPE_QL_TYPE, NULL);
```
3. Create a DDO using the `DKDatastoreICM`’s `createDDO` method. Do not use the `DKDDO` constructor. In the example below, the object `dsICM` is already connected to a Content Manager datastore. Also, the PID is a string named `pidStr`.

```
DKDDO* ddo = dsICM->createDDO(pidStr);
```
4. Call the DDO’s retrieve operation.

```
ddo->retrieve();
```

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

Example:

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

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

Example:

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

For more information on retrieving items, see the `SItemRetrievalICM` and `SResourceItemRetrievalICM` ICM API education samples and samples readme file.

If an item is enabled for versioning and you want to retrieve the latest version, including its attributes and children, use the following format:

Java

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

C++

```
DKDDO * ddo;
long options =
DK_CM_CONTENT_ATTRONLY +DK_CM_CONTENT_CHILDREN +DK_CM_VERSION_LATEST;
dsICM->retrieveObject(ddo,options);
```

When retrieving items, the system default is to retrieve items using uncommitted reads. Prior to DB2 Content Manager Version 8.3, the DB2 Content Manager system used committed reads when retrieving an item. The exposure for performing an uncommitted read in this situation is very small. An example of an uncommitted read is where a user might view partially committed data for a document in a multi-level item type, and then only if the retrieval is done while another update to the exact same item is in progress by another user.

If you want to revert back to behavior that was available before fix pack 7 when performing retrieve operations, follow one of the two following methods:

- Use the new API flag

`DK_CM_CONTENT_RETRIEVE_USING_COMMITTED_READ`

You can use the API flag `DK_CM_CONTENT_RETRIEVE_USING_COMMITTED_READ` with the retrieve APIs to indicate that committed reads should be performed by the library server when retrieving an item. If this flag is passed (as a bitmask) as part of the option argument, the library server performs committed reads when performing the retrieve operation. By using this option, applications can get the same behavior that is available before DB2 Content Manager Version 8.2 fix pack 7 when performing retrieve operations.

- Set the environment variable `ICM_RETRIEVE_OPTION`

Set the environment variable ICM_RETRIEVE_OPTION on the library server machine. On the Windows operating system, set the ICM_RETRIEVE_OPTION as an environment variable. On UNIX operating systems, set the ICM_RETRIEVE_OPTION in the <instance_home>/sqllib/profile.env and userprofile file in the same manner as other DB2 Content Manager variables, such as ICMROOT.

A new flag called DK_CM_CONTENT_RETRIEVE_IGNORE_ACL_NAME is available with the retrieve APIs. If this flag is passed (as a bitmask), as part of the option argument, the retrieve APIs will not attempt to fetch the ACL name associated with the item being retrieved. The DK_ICM_PROPERTY_ACL property associated with the retrieved DDO object will be an empty string. Using this option can improve performance in situations where the ACL name associated with an item is not important to an application. The performance benefit is applicable only in situations where the ACL name has already not been fetched and cached. Not having to fetch the ACL name (if it already is not cached) will reduce a stored procedure call to the library server. Applications can fetch the ACL name associated with an item (at a later time, if you want) by using the projection mechanism to retrieve system attributes

For more information about retrieve, such as retrieval options, see the SItemRetrievalICM sample.

Deleting items

Use the delete method in the DDO to delete an item from the content server.

Java

```
ddoDocument.del();
```

C++

```
ddoDocument->del();
```

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

For more information about deleting items, see the SItemDeletionICM ICM API education sample.

Deleting versioned items

If you enabled an item for versioning, remember that you have various versions of that item that you can work with. If you want to completely delete a versioned item, you must delete all of the item's versions. This section describes how to delete all versions of an item or individual versions of a versioned item.

To delete individual versions of an item, search for all of the items matching some search criteria, such as the ITEMID. If you don't specify a version or the latest version, all versions matching the search criteria are returned. If ITEMID is the only criteria you specify, all versions of that item are returned. You can then loop through the results collection and delete each item individually.

To delete all versions of an item at once, specify the DKConstantICM::DK_ICM_DELETE_ALL_VERSIONS option when deleting any individual item version. It is important to specify the

DKConstantICM::DK_ICM_DELETE_ALL_VERSIONS because if you do not, only the item version represented by the root component DDO is deleted.

To see example code that uses search and delete, see the SItemDeletionICM sample in the samples directory.

Checking in and checking out items

To prevent two users from making changes to the same item at the same time, you must check out an item before updating it. Checking out an item grants exclusive update rights to the user who has the item checked out. When you check out an item you hold a persistent write lock on that item. The entire item is locked and unlocked as a whole, including all versions and child components. Linked and referenced items, however, are not locked with the item. The persistent lock on the item is released when you check in the item.

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

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

Java

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

C++

```
dsICM->checkOut(myDataObject);
```

2. Check in the item.

Java

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

C++

```
dsICM->checkIn(myDataObject);
```

For more information on checking items in and out, see the SItemUpdateICM ICM API education sample.

Setting and getting an item's versioning properties

The example below, demonstrates how to set an item's versioning properties and how to retrieve and item's versioning properties.

Java

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();
// Accessing version information
String version = pid.getVersionNumber();
...
// Setting the version number in the DDO
pid.setVersionNumber(version);
```

C++

```
DKPidICM * pid = (DKPidICM *)ddo->getPidObject();
// Accessing version information
DKString version = pid->getVersionNumber();
....
//Setting the version number for the PID associated with an item (DDO).
pid->setVersionNumber((char *)version);
```

Working with item versions

You can keep multiple versions of items and objects by enabling the versioning options provided by DB2 Content Manager. For example, if an item has the versioning option set to *always*, a new version is created each time the item is updated instead of updating the original or existing version. There are things that you can do to ensure that you set the version policy that is best meets your needs. For example, you can turn off version control for attributes, and leave the version control for resource content. You can also allow an end user application to specify when a new version is created.

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

Retrieving the version control policy of an item type

An item has a version control policy, which contains the versioning property for the item. Following is the list of the three versioning properties available and the value used to represent each property in the version control policy.

DK_ICM_VERSION_CONTROL_ALWAYS

Versioned-always.

DK_ICM_VERSION_CONTROL_NEVER

Versioning is not supported for this item type (default).

DK_ICM_VERSION_CONTROL_BY_APPLICATION

The application determines the versioning scheme.

Java

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

C++

```
short versionControlPolicy = 0;
DKItemTypeDefICM * item    = NULL;
...
versionControlPolicy = item->getVersionControl();
```

Setting version control for an item type

Java

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

C+

```
DKItemTypeDefICM * item = NULL;
short versionControl    = DK_ICM_VERSION_CONTROL_ALWAYS;
...
item->setVersionControl(versionControl);
```

For a complete sample that accesses version control information of item type definitions in the system, see the `SItemTypeRetrievalICM` ICM API education sample.

Getting the maximum number of versions allowed for an item type

Java

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

C++

```
short versionMax    = 0;
DKItemTypeDefICM * item = NULL;
....
versionMax = item->getVersionMax();
```

Setting the maximum number of versions allowed for an item type

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

Java

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


C++

```
short versionMax    = 10;
DKItemTypeDefICM * item = NULL;
...
item->setVersionMax(versionMax);
```

Setting the value of the versioning type for an item type

C++

```
DKItemTypeDefICM * item = NULL;
short versionType = DK_ICM_ITEM_VERSIONING_OPTIMIZED;
...
item->setVersioningType(versioningType);
```

Working with folders

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

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

Creating a folder

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

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

Java

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

C++

```
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);
```

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

Java

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

C++

```
ddoFolder->add();
```

Adding contents to a folder

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

To add items to a folder, use the DKFolder's addElement() function. When you have added enough members into the folder, you can call the add or update method to save the folder-content relationships into the persistent store. This groups any number of folder modifications and into a single call to the library server. When adding items to a folder, do not add multiple copies of the same item to DKFolder. Since all folder modifications are tracked, do not cause conflicting or duplicate modifications. For example, do not add multiple copies of the same item to the folder.

Follow the steps below to add contents to a folder. Note that a folder can contain another folder (sub-folder) as one of its content items.

Java

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

```
DKDDO ddoDocument=dsICM.createDDO("S_simple",DKConstant.DK_CM_DOCUMENT);  
DKDDO ddoFolder2=dsICM.createDDO("S_simple",DKConstant.DK_CM_FOLDER);  
DKDDO ddoItem=dsICM.createDDO("S_simple",DKConstant.DK_CM_ITEM);
```
2. Use the setData method to populate the DDOs. Items that are to be added as folder contents need to be persisted into the datastore.

```
ddoDocument.add();  
ddoItem.add();  
ddoFolder2.add();
```
3. Retrieve the dkFolder attribute of the previously created and persisted folder DDO.

```
short dataid = ddoFolder.dataId  
    (DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKFOLDER);  
dkFolder = (DKFolder) ddoFolder.getData(dataid);
```
4. The folder must be checked out of the datastore before it is updated (by adding contents).

```
dsICM.checkOut(ddoFolder);
```
5. Add the previously created items to the folder.

```
dkFolder.addElement(ddoDocument);  
dkFolder.addElement(ddoItem);  
dkFolder.addElement(ddoFolder2);
```
6. Commit the changes to the folder and explicitly check in the changes.

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

C++

1. Create the items that will be added to the folder.

```
DKDDO * ddoDocument = dsICM->createDDO("book", DK_CM_DOCUMENT);  
DKDDO * ddoFolder2   = dsICM->createDDO("book", DK_CM_FOLDER);  
DKDDO * ddoItem       = dsICM->createDDO("book", DK_CM_ITEM);
```
2. Persist the created items to the datastore.

```
ddoDocument->add();  
ddoItem->add();  
ddoFolder2->add();
```
3. Retrieve the DKFolder attribute for the folder.

```
DKFolder * dkFolder;  
short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKFOLDER);  
if (dataid!=0) dkFolder =  
    (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
```
4. Check out the folder (A folder must be checked out before it is update)

```
dsICM->checkOut(ddoFolder);
```
5. Add the created items to the folder.

```
dkFolder->addElement(ddoDocument);  
dkFolder->addElement(ddoItem);  
dkFolder->addElement(ddoFolder2); // Folders can contain sub-folders.
```
6. Update the folder. This implicitly also checks in the folder (unlocks it).

```
ddoFolder->update();
```
7. Explicitly check in the folder.

```
dsICM->checkIn(ddoFolder);
```

Removing contents from a folder

Items can be removed from a folder in one of two ways. A deferred removal (the actual removal is done when the folder DDO is updated and multiple content server calls are combined into one) is done using the `removeElementXX` method(s). An immediate (and expensive since multiple calls are made to the content server) removal can be done using the `dkFolder.removeMember` method. See the `SFolderICM` sample (in the samples directory) for further details on working with folders.

Complete the following steps to remove contents from a folder:

Java

1. Check out the folder DDO from which we want to remove an item.
`dsICM.checkOut(ddoFolder);`
2. Look for the item to remove. In this case, look for the docItem DDO created earlier. Create an iterator to iterate through the folder's content.

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

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

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

C++

1. Create an item and add it to a folder. Note that the folder was created earlier.

```
DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);
ddoItem->add();
DKFolder * dkFolder;
short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKFOLDER);
if (dataid!=0) dkFolder =
    (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
dsICM->checkOut(ddoFolder);
dkFolder->addElement(ddoItem);
ddoFolder->update();
```

2. Explicitly check in the folder.

```
dsICM->checkIn(ddoFolder);
```

3. Create an iterator for the folder.

```
dkIterator* iter = dkFolder->createIterator();
```

4. Iterate through the contents of the folder until the item that is to be removed is found. Remove the item.

```
while(iter->more())
{
    DKDDO* ddo = (DKDDO*) iter->next()->value();
    if ( ((DKPidICM*) ddo->getPidObject())->pidString() ==
        ((DKPidICM*) ddoItem->getPidObject())->pidString() )
    {
        //Found the element to be removed. Remove it
        dkFolder->removeElementAt(*iter);
        // Now that we found the ddoItem, remove it.
    }
}

delete(iter);
```

Tips for adding and removing folder contents

You can streamline the process for adding, updating, and removing folder contents by remembering the following points:

Use element methods to group multiple updates into a single update

The most efficient way to add or remove any amount of folder contents is to group all additions and modifications into a single update to the library server. Use the DKFolder add element or remove element functions. When all modifications are complete, call the DKDDO's add() or update() method.

Use single, periodic create and delete because immediate methods provide the best performance

If you add or remove items one at a time without making any other changes to the folder, the immediate methods work best because they avoid the additional operations that occur in a DKDDO::update(). For immediate results, use the DKFolder.addMember() or removeMember() or DKDatastoreExtICM's addToFolder() or removeFromFolder() functions, which make the changes persistent immediately at the cost of one call to the library server for every change. However, this becomes time consuming if you add or delete more than one item.

Retrieving a folder's contents

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

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND
- DKConstant.DK_CM_CONTENT_ITEMTREE

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

Java

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

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_LINKS_OUTBOUND);
```

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

Java

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

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_ITEMTREE);
```

Obtaining all folders containing a specific DDO

Multiple folders can contain the same item (DDO). This allows you to associate the item with different applications or users. To determine which folders currently contain a specific DDO, see the example below.

The following steps demonstrate how to find folders that contain a `ddoDocument` object that was previously created. See the `SFolderICM` sample (in the samples directory) for additional details about working with folders.

Java

1. Retrieve the datastore extension object.

```
DKDatastoreExtICM dsExtICM =(DKDatastoreExtICM)
                        dsICM.getExtension(DKConstant.DK_CM_DATASTORE_EXT);
```
2. Call the `getFoldersContainingDDO` method on the extension object to find the folders containing the `ddoDocument` object. This returns a collection of DDOs where each returned DDO is a folder containing the `ddoDocument` object.

```
DKSequentialCollection list =
    dsExtICM.getFoldersContainingDDO(ddoDocument);
```
3. Create an iterator to cycle through the returned collection of folders.

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

C++

```
DKDDO* ddoDocument = ....;
//Create datastore object,connect to datastore,create DDO etc.
....
//Create a new datastore extension object from the connected object
DKDatastoreExtICM* dsExtICM = (DKDatastoreExtICM*)
                                dsICM->getExtension(DK_CM_DATASTORE_EXT);
//Retrieve the PID for the created DDO
DKPidICM* pid = (DKPidICM*) ddoDocument->getPidObject();
//Retrieve a list of folders containing the DDO created earlier.
DKSequentialCollection *list =
    dsExtICM->getFoldersContainingDDO(ddoDocument);
//Create a iterator for the returned DDO collection
dkIterator* iter =list->createIterator();
while(iter->more()) {
    DKDDO* ddo =(DKDDO*) iter->next()->value();
    pid = (DKPidICM*) ddo->getPidObject();
    cout << "-Item ID:" << pid->getItemId() << endl;
    delete ddo;
}
delete iter;
```

Defining links between items

You can use links to associate a source item to a target item with an optional description item. You define links in `DKLink` objects, where you can specify the

following types of link elements:

Table 14. Link data structure

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

Since a link represents a one to many relationship, it is represented in a DDO by a data-item under LINK namespace, of value DKLinkCollection. For more information about working with links, see the SLinksICM sample in the samples directory.

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

In memory, both the source and the target DDOs contain copies of the same DKLink object. Since the DKLink object contains a reference to both the source and the target, a DDO containing a link also contains a link that refers to itself as well as to another DDO. For example, if Source A is linked to Target B, both A and B contain the same link, as shown in the example in Table 15.

Table 15. DKLink definition example

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

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

For more information about links and a complete links sample that you can run, see the SLinksICM sample in the samples directory.

Inbound and outbound links

When using links to reference a particular DDO, either the source or the target, you can consider the links to be *inbound* or *outbound*. When considering a link from the perspective from a particular DDO, if that DDO is the target, then that DDO considers it an inbound link. Meanwhile, the DDO at the other end of that link considers the same link to be outbound from its perspective.

In the example in Table 15, the link from DDO A to DDO B is outbound link, while the same link to DDO B is the inbound link.

Link type names

Link relationships have names. Within a DDO, links are grouped into link collections. There are system defined link type names and you can also define your own. You can use any number of user-defined link type names or system-defined link type names. The following system defined link type names include:

Link type name contains

Java Constant: DKConstant.DK_ICM_LINKTYPENAME_CONTAINS

C++ Constant: DK_ICM_LINKTYPENAME_CONTAINS

Link type name: DKFolder

Java Constant : DKConstant.DK_ICM_LINKTYPENAME_DKFOLDER

C++ Constant : DK_ICM_LINKTYPENAME_DKFOLDER

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

See the SLinksICM sample, in the IBMCMROOT/samples/cpp/icmor IBMCMROOT/samples/java/icmdirectory, for additional information about links. For information on creating user-defined link types, see the SLinkTypeDefinitionCreationICM sample.

Retrieving linked items

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

Java

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

C++

- DK_CM_CONTENT_LINKS_OUTBOUND
- DK_CM_CONTENT_LINKS_INBOUND
- DK_CM_CONTENT_LINKS_OUTBOUND + DK_CM_CONTENT_LINKS_INBOUND
- DK_CM_CONTENT_ITEMTREE

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

Working with access control

If you have access to either the DB2 Content Manager system administration client or to the DB2 Content Manager APIs for writing your own administration program, you can use the access control functions to control access to the information in your DB2 Content Manager system. The various access control APIs allow you to control access to the entire system, to a closely related set of operations on the system, or to an individual item.

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

- Creating privileges.
- Associate a list of actions with information in the system.

- Give permission to users to perform actions on the information in the library server.

Creating a privilege

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

Java

1. Connect to a datastore

```
DKDatastoreICM ds = new DKDatastoreICM();
ds.connect("ICMNLSDDB","icmadmin",password,"");
dkDatastoreDef dsDef =(dkDatastoreDef)ds.datastoreDef();
DKDatastoreAdminICM dsAdmin =(DKDatastoreAdminICM)dsDef.datastoreAdmin();
```
2. Retrieve a `DKAuthorizationMgmtICM` object. This class handles authorization management tasks.

```
DKAuthorizationMgmtICM aclMgmt = (DKAuthorizationMgmtICM)
dsAdmin.authorizationMgmt();
```
3. Create a new privilege object.

```
DKPrivilegeICM priv = new DKPrivilegeICM(ds);
```
4. Assign a name to the privilege object.

```
priv.setName("UserPriv");
```
5. Assign a description to the privilege object.

```
priv.setDescription("This is user-defined privilege");
```
6. Add the new privilege to the authorization manager object.

```
aclMgmt.add(priv);
```
7. Retrieve the newly created privilege from the authorization manager object.

```
dkPrivilege aPriv = aclMgmt.retrievePrivilege("UserPriv");
```
8. Display information about the privilege object.

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

C++

1. Create the datastore object as well as all the related administration management objects.

```
//Create the datastore object
DKDatastoreICM * ds = new DKDatastoreICM();
//Connect to the underlying datastore
ds->connect("ICMNLSDDB", "icmdmin", "password", "");
//Retrieve the datastore definition object (used to access
//and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore
//administration functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
    dsAdmin->authorizationMgmt();
```

2. Create a new privilege object and set its properties

```
DKPrivilegeICM * priv = new DKPrivilegeICM(ds);
//Set the name of the privilege object
priv->setName("UserPriv");
//Set the privilege description
priv->setDescription("This is user-defined privilege");
```

3. Add the privilege to the datastore via the authorization management object.

```
aclMgmt->add(priv);
```

Creating a privilege set

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

Java

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

```
DKPrivilegeICM priv_1 = new DKPrivilegeICM(ds);
priv_1.setName("ItemCheckOut");
DKPrivilegeICM priv_2 = new DKPrivilegeICM(ds);
priv_2.setName("ItemQuery");
DKPrivilegeICM priv_3 = new DKPrivilegeICM(ds);
priv_3.setName("ItemAdd");
```
2. Create a new privilege set.

```
DKPrivilegeSetICM privSet1 = new DKPrivilegeSetICM(ds);
```
3. Assign a name to the privilege set.

```
privSet1.setName("UserPrivSet");
```
4. Assign a description to the privilege set.

```
privSet1.setDescription("This is a user-defined priv set");
```
5. Add the privileges to the privilege set.

```
privSet1.addPrivilege(priv_1);
privSet1.addPrivilege(priv_2);
privSet1.addPrivilege(priv_3);
```
6. Add the newly created privilege set to the authorization manager.

```
AclMgmt.add(privSet1);
```
7. Display information about the newly created privilege set.

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

C++

1. Create the datastore object as well as all the related administration management objects.

```
//Create the datastore object
DKDatastoreICM * ds = new DKDatastoreICM();
//Connect to the underlying datastore
ds->connect("ICMNLSDb", "icmdmin", "password", "");
//Retrieve the datastore definition object
//(used to access and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore administration
// functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
    dsAdmin->authorizationMgmt();
```

2. Create three privileges and set their properties.

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

3. Create a new privilege set and set its properties.

```
DKPrivilegeSetICM * privSet1 = new DKPrivilegeSetICM(ds);
privSet1->setName("UserPrivSet");
privSet1->setDescription("This is a user-defined priv set");
```

4. Add the created privileges to the new privilege set.

```
privSet1->addPrivilege(priv_1);
privSet1->addPrivilege(priv_2);
privSet1->addPrivilege(priv_3);
```

5. Add the privilege set to the datastore using the authorization management object.

```
aclMgmt->add(privSet1);
```

Displaying privilege set properties

The example below demonstrates how to display a privilege set's properties.

C++

```
//Retrieve a privilege set using its name
DKPrivilegeSetICM * sPrivSet = (DKPrivilegeSetICM *)
    aclMgmt->retrievePrivilegeSet("UserPrivSet");
//Display privilege set properties
cout<<"privilege set name = "<< (char *)sPrivSet->getName() << endl;
cout<<"priv set descrip="<< (char *)sPrivSet->getDescription() << endl;
//Retrieve the list of privileges that are a part of this privilege set
dkCollection * coll = sPrivSet->listPrivileges();
dkIterator * iter = coll->createIterator();
while (iter->more())
{
    DKPrivilegeICM* _priv = (DKPrivilegeICM *) (void *) (*iter->next());
    cout<<"  privilege name = "<< (char *)_priv->getName() << endl;
}
delete(iter);
```

Defining an access control list (ACL)

The DB2 Content Manager access control model is applied at the level of the controlled entity. A controlled entity is a unit of protected user data. A controlled entity can be an individual item, item-type, or the entire library. Operations on controlled entities are regulated by one or more control rules. The ACL is the container for these control rules. The `DKAccessControlListICM` class represents an ACL. Every controlled entity in a DB2 Content Manager system must be bound to an ACL.

The examples below demonstrate how to define an ACL.

Java

```
//Define a new DKACLData object.
//A DKACLData class is used to hold ACL related data.
DKACLData aclData1 = new DKACLData();
//set the user group name for the ACL
aclData1.setUserGroupName("ICMADMIN");
//set ACL patron type.
aclData1.setPatronType(DK_CM_USER_KIND_USER);
//Set the privilege set associated with this ACL
aclData1.setPrivilegeSet(privSet_1);
//Create another DKACLData object.
DKACLData aclData2 = new DKACLData();
aclData2.setUserGroupName("ICMPUBLIC");
aclData2.setPatronType(DK_CM_USER_KIND_GROUP);
aclData2.setPrivilegeSet(privSet_2);
//Create a new ACL. A DKAccessControlListICM represents a CM v8.3. ACL
DKAccessControlListICM acl1 = new DKAccessControlListICM(ds);
//Assign a new to the newly-created ACL
acl1.setName("UserACL");
//Assign a description to the newly-created ACL
acl1.setDescription("This is a user-defined ACL");
//Add the previously created ACL data objects to the ACL
acl1.addACLData(aclData1);
acl1.addACLData(aclData2);
//Add the newly-created ACL to the authorization manager
aclMgmt.add(acl1);
//Retrieve and display information about the newly created ACL
DKAccessControlListICM acl_1 = (DKAccessControlListICM)
aclMgmt.retrieveAccessControlList("UserACL");
System.out.println("ACL name = " + acl_1.getName());
System.out.println("    desc = " + acl_1.getDescription());
dkCollection coll = acl_1.listACLData();
dkIterator iter = coll.createIterator();
while (iter.more()) {
    DKACLData aclData = (DKACLData) iter.next();
    DKPrivilegeSetICM _privSet = (DKPrivilegeSetICM) aclData.getPrivilegeSet();
    System.out.println("  PrivSet name = " + _privSet.getName());
    System.out.println("  PrivSet desc = " + _privSet.getDescription());
    String usrGrpName = aclData.getUserGroupName();
    System.out.println("    UserGroupName = " + usrGrpName);
    short patronType = aclData.getPatronType();
    System.out.println("    Patron type = " + patronType);
}
```

C++

1. Define new DKACLData objects. Each DKACLData object is used to hold ACL related data.

```
DKACLData * aclData1 = new DKACLData();
// set the name of the user group to be associated with this ACL
aclData1->setUserGroupName("ICMADMIN");
// Set the ACL patron type. Can be one of 3 possible
//values:DK_CM_USER_KIND_USER or DK_CM_USER_KIND_GROUP or
//DK_CM_USER_KIND_PUBLIC.
aclData1->setPatronType(DK_CM_USER_KIND_USER);
// Set the privilege set associated with the ACL.
DKPrivilegeSetICM * privSet_1 = (DKPrivilegeSetICM *)
    aclMgmt->retrievePrivilegeSet("UserPrivSet");
aclData1->setPrivilegeSet(privSet_1);
```

2. Create another DKACLdata object.

```
DKACLData * aclData2 = new DKACLData();
aclData2->setUserGroupName("ICMPUBLIC");
aclData2->setPatronType(DK_CM_USER_KIND_GROUP);
DKPrivilegeSetICM * privSet_2 = (DKPrivilegeSetICM *)
    aclMgmt->retrievePrivilegeSet("PublicPrivSet");
aclData2->setPrivilegeSet(privSet_2);
```

3. Create a new ACL. Set property values for this new ACL.

```
DKAccessControlListICM * acl1 = new DKAccessControlListICM(ds);
//Assign a new name to the newly-created ACL.
acl1->setName("UserACL");
// Assign a description to the newly-created ACL.
acl1->setDescription("This is a user-defined ACL");
```

4. Add the ACL, created in step three, data objects to the ACL.

```
acl1->addACLData(aclData1);
acl1->addACLData(aclData2);
```

5. Add the ACL, created in step three, to the authorization manager.

```
aclMgmt->add(acl1);
```

The complete sample program is in the samples directory.

Retrieving and displaying ACL information

To retrieve and display ACL information, complete the following steps.

1. Retrieve the ACL from the authorization management object using the ACL's name.

C++

```
DKAccessControlListICM * acl_1 = (DKAccessControlListICM *) aclMgmt->
    retrieveAccessControlList("UserACL");
cout<<"ACL name = "<< (char *)acl_1->getName() << endl;
cout<<"    desc = "<< (char *)acl_1->getDescription() << endl;
```

2. Retrieve the ACL data associated with the ACL.

C++

```
dkCollection * coll = acl_1->listACLData();
dkIterator * iter = coll->createIterator();
char szpatronType[12] = {0x00};
while (iter->more())
{
    DKACLData * aclData = (DKACLData *) (void *) (*iter->next());
    DKPrivilegeSetICM * _privSet = (DKPrivilegeSetICM *) aclData->
        getPrivilegeSet();
    cout<<"privSet name = "<< (char *) _privSet->getName() << endl;
    cout<<"privSet desc="<< (char *) _privSet->getDescription() << endl;
    DKString usrGrpName = aclData->getUserGroupName();
    cout<<"    UserGroupName = "<< (char *) usrGrpName << endl;
    short patronType = aclData->getPatronType();
    sprintf(szpatronType, "%d", patronType);
    cout<<"    Patron type    = "<< szpatronType << endl;
}
delete(iter);
```

Assigning an ACL to an item type

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

1. Set up anew item type.

Java

```
DKItemTypeDefICM it = new DKItemTypeDefICM(dsICM);
//Assign a name to this item type
it.setName("TextResource1");
//Assign a description to this item type
it.setDescription("CMv8.3 Text Resource Item Type.");
//Assign an ACL code to this item type
it.setItemTypeACLCode((int)DK_ICM_ITEMACL_BIND_AT_ITEM);
....
it.add(); // make the item type persistent
...
```

C++

```
DKItemTypeDefICM * itemType = new DKItemTypeDefICM(dsICM);
// Assign a name to this item type.
itemType->setName("TextResource1");
// Assign a description to this item type.
itemType->setDescription("CMv8.3 Text Resource Item Type.");
```

2. Retrieve the ACL data associated with the ACL.

Java

```
int itemTypeACLCode = it.getItemTypeACLCode();
// By setting the item type's ACL flag to TRUE(1) we confirm that ACL
// binding is at the item type level. By setting the item type's ACL
// flag was set to FALSE(0), we would be saying that the ACL binding
// is not at the item type level
it.setItemLevelACLFlag(1); // - true
//Determine whether the ACL binding is at the item type level or not
int itemTypeACLFlag = it.getItemLevelACLFlag();
```

C++

```
itemType->setItemTypeACLCode((long) 1);
// By setting the item type's ACL flag to TRUE (1),
//we confirm that ACL binding is at the item type level.
//By setting the item type's ACL flag to FALSE(0),
// we would be saying that the ACL binding is not at the item type level.
itemType->setItemLevelACLFlag((short)1);
itemType->add();
// make the item type persistent.
```

The complete sample program is available in the samples directory.

Assigning an ACL to an item

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

Java

```
// Assume that an ACL (Access Control List) named "MyACL"
// already exists in the system. Add a new property to the DDO.
//This property will be called DK_ICM_PROPERTY_ACL
int propId = ddoItem.addProperty(DK_ICM_PROPERTY_ACL);
//Set the previously created (or retrieved)ACL as the value of this property
ddoItem.setProperty(propId,"MyACL");
//Persist the DDO into the data store
ddoItem.add();
```

C++

1. Create a new item (DDO) based on an existing item type.
`DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);`
2. Assume that an ACL (Access Control List) with name "MyACL" already exists in the system.
3. Add a new property to the DDO. This property will be called DK_ICM_PROPERTY_ACL.
`ushort propId = ddoItem->addProperty(DK_ICM_PROPERTY_ACL);`
4. 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");`
5. Persist the DDO into the data store.
`ddoItem->add();`

Library server and federated database limitations

When you connect to a library server using a DB2 Content Manager or federated database user ID that is not a database user ID, the connector will try to connect to the database using that user ID and the connection will fail. The connector will then connect using the database connect user ID, and pass the DB2 Content Manager or federated database user ID to the server.

To prevent the database connection from failing when using a DB2 Content Manager or federated database user ID which is not a database user ID, you can do the following:

- **For a federated database:**

Update the FEDSERVERREPTYPE value to DB2CON instead of DB2 for a particular federated database in the cmbds.ini file. It should look like this:
FEDSERVERREPTYPE=DB2CON.

Modifying the FEDSERVERREPTYPE to DB2CON will cause the connector to log on using the database connect ID first, and the connector will then pass the federated database user ID to the server for that federated database. If the federated database user ID is also a database user ID, the call to the server will fail. The connector will automatically disconnect from the database and reconnect using the federated user ID to log on to the database and then pass the federated database user ID to the server.

For the Java and C++ DKDatastoreFed connector connect method, there is another connect string option that you can specify so that a particular instance of the datastore has the behavior mentioned above. The option is REPTYPE=DB2CON

- **For a DB2 Content Manager Version 8 database:**

Update the ICMSEVERREPTYPE value to DB2CON instead of DB2 for a particular DB2 Content Manager Version 8 database in the cmbicmsrvs.ini file, like this: ICMSEVERREPTYPE=DB2CON

Once you change the ICMSEVERREPTYPE value, the DB2 Content Manager connector will log on to the server using the database connect ID first, and then pass the DB2 Content Manager Version 8 user ID to the server for that DB2 Content Manager Version 8 database. If the DB2 Content Manager Version 8 user ID is also a database user ID, the call to the server will fail. The connector will

automatically disconnect from the database and reconnect using the DB2 Content Manager Version 8 user ID to log on to the database, and then pass the DB2 Content Manager Version 8 user ID to the server.

For the Java and C++ DKDatastoreICM connector connect method, you can specify a connect string option, so that a particular instance of the datastore has the behavior mentioned above. The option is

REPTYPE=DB2CON

Working with the resource manager

A DB2 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 DB2 Content Manager system provides a common subset of native data access APIs through which it is accessible by the controlling library server, by other DB2 Content Manager components, and by applications, either locally (on the same network node) or remotely.

Other data access APIs allow remote access to a resource manager using the resource manager's own client support or a standard network access protocol such as CIFS, NFS, or FTP. For remote access, use a client-server connection. Clients communicate with Content Manager resource managers using HTTP through the use of a standard Web server. Data delivery is based on HTTP, FTP, and FS data transfer protocols. Using HTTP, any application or Content Manager component that needs to access Content Manager-managed content can dynamically form a triangle with a library server and resource manager. This triangle forms a direct data access path between the application and each resource manager, and a control path between the library server and the resource manager. You can map this conceptual triangle to any network configuration, ranging from a single-node configuration to a geographically distributed one.

The new architecture also accommodates resource managers that an application is not able to access directly, such as a host-based subsystem, a single-user system that does not handle access control, or a system containing highly sensitive information where direct access by an application is not allowed by business policy. In this case, access to such resource managers is indirect. Both the pull and the push paradigms of data transfer are accommodated by the Content Manager system as well as synchronous and asynchronous calls.

For information about how to configure a resource manager see *Planning and Installing Your Content Management System* and the SResourceMgrDefCreationICM sample in the samples directory.

Working with resource manager objects

Within DB2 Content Manager, every managed entity is called an item. Items come in two types, the type that represent pure logical entities, such as documents or folders, or entities that represent physical data objects, such as the text data of a word processing document, the scanned image of a claim or the video clip of an automobile accident. Objects have a special state and behavior needed to handle the physical data associated to a logical document.

Resource objects also represent things like files in a file system, video clips in a video server, and BLOBs. At run time, resource objects are used to access the physical data they point to. For that reason, resource objects in Content Manager have a type. That is, they have a specific state and behavior. The library server and the resource manager share a schema to store the state of an object. The base object types provided by Content Manager are: generic BLOBs or CLOBs, Text, Image, and Video content objects. You can also create sub-classes of the pre-defined types. A resource object can also have user-defined attributes, which are used for search and retrieval.

From the Content Manager system perspective, each object is represented by a unique logical identifier, its Uniform Resource Identifier (URI). The library server manages the URI name space. On request, the library server maps URIs onto Uniform Resource Locators (URL). URLs are used to gain access to the physical data. URLs do not point directly to a storage area managed by the resource manager. Instead, the resource manager uses a local name space to convert logical object names to physical file names. Object URIs are created by the specific resource manager. The library server or the end-user can suggest an object URI (its name), but the decision is made by the resource manager.

You can access an object using the DB2 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, `IBMCMROOT/samples/java/icm` or `IBMCMROOT/samples/cpp/icm`.

Confidential retrieval of resource objects

DB2 Content Manager supports Secure Sockets Layer (SSL) through the Java APIs for communications between thin client or applet applications and the resource manager. You should use SSL if your application accesses resource manager objects and runs outside of a secure boundary, like a firewall.

Important: Before you use your application to communicate with the resource manager through SSL, ensure that the HTTP server on the resource manager is enabled for SSL. For the eClient viewer applet, ensure that the eClient Server is configured to use an HTTPS connection.

When you call the `DKLobICM.getContentURLs` Java API within your application, it returns a URL. If the URL returned by `DKLobICM.getContentURLs` is an HTTPS URL, SSL is enabled. If the URL is an HTTP URL, SSL is not enabled. To enable SSL, go to the `cmbrm.ini` file, which you can find in the same directory as other properties files. You can also find the `cmbrm.ini` file by checking the `cmbrmenv.properties` file, which contains an entry called `CMCFGDIR` that points to the location of `cmbrm.ini` file. Open the `cmbrm.ini` file and set the `RM_SSL_FOR_URL_RETRIEVE` setting to 1. If the setting does not exist, add it.

Once you have verified that the `RM_SSL_FOR_URL_RETRIEVE` setting is correct, your application can use the HTTPS URL to securely communicate with the resource manager (or LAN cache). Note that `RM_SSL_FOR_URL_RETRIEVES` is a global setting, and that once it is set to 1, all URL-based retrieves with all resource managers in the system will use SSL. That means that if your system is setup to work with multiple resource managers, all the resource managers must support SSL.

Removing resource object contents

When you delete the content associated with a resource item, you can irrecoverably destroy that content. This function, called irrecoverable destroy, is only supported in the DB2 Content Manager Version 8 connector for objects stored on a fixed disk within a multiplatform (Windows, AIX, Solaris) resource manager. To use this function, the DB2 Content Manager Version 8 connector, library server, and resource manager must be at the DB2 Content Manager Version 8.2 fix pack level seven or later. If any of the DB2 Content Manager components are at previous level, irrecoverable destroy is processed as a normal delete.

To use irrecoverable destroy, you must specify the DKConstant option `DK_CM_DESTROY_DELETE` in operations that result in the deletion of an object. Following is a list of the operations that delete objects, and where you can specify irrecoverable destroy:

- When calling delete on a resource item or document.
- When calling update to delete a document part or when updating a resource item from content to no-content.
- When calling update for a resource item that has versioning enabled. In this case, when the maximum number of stored versions is exceeded, and the earliest version is to be deleted, the earliest version is also destroyed.
- When an item is being reindexed into a new item type, the item from the original item type is destroyed.

See the Application Programming Reference for detailed information about the methods and options you work with to delete objects. Following is a list that summarizes the APIs and associated methods where you can specify irrecoverable destroy:

- **DKDatastoreICM:**

```
updateObject(dkDataObject ddo, int option)
deleteObject(dkDataObject ddo, int option)
moveObject(dkDataObject sourceDDO, dkDataObject destinationDDO, int
options)
```

- **DKLobICM:**

```
del(int option)
update(InputStream is,long length,int option)
update(DKThirdPartyServerDef thirdpartyObject, int option)
update(int option)
update(String aFullFileName,int option)
update(int option,DKRMSMPairDefICM[] rmsmpsairs)
updateFrom( String hostname, String userid, String passwd, String
protocol, int port, String filename,int option)
updateFrom(int option)
updateFromAsync( String hostname, String userid, String passwd, String
protocol, int port, String filename,int option)
updateFromAsync(int option)
```

- **DKDDO:**

```
update(int option)
update(DKNVPair[] option)
del(int option)
```

del(DKNVPair[] option)

In most cases you can combine DK_CM_DESTROY_DELETE with other options, except in the case of a delete. For delete operations, this flag must be specified exclusively. When the content associated with all versions of a resource item must be destroyed, you must substitute the DK_CM_DESTROY_DELETE option with the DK_ICM_DESTROY_ALL_VERSIONS option. The option DK_ICM_DESTROY_ALL_VERSIONS is available only for delete.

Understanding asynchronous replication in a z/OS resource manager

Asynchronous replication copies an object from one DB2 Content Manager resource manager to another. For objects that you want to replicate, You must create replication rules and associate the rules with the collection where the objects will be stored. You must do this before storing objects into DB2 Content Manager. In the event objects were stored to DB2 Content Manager before the creation and association of replication rules, DB2 Content Manager provides a utility called IMPORTREPLICA for this purpose.

Asynchronous replication functionality is encapsulated in a standalone module of the z/OS resource manager. Its current implementation is asynchronous. To complete an asynchronous replication process, complete the following steps:

1. Run the ICMRMAP.JCL to start the asynchronous replication process, ICMOSAP. The input parameter list passed into ICMOSAP from ICMRMAP.JCL is then verified. The input parameter list is as follows:

DB2SUBSYSID (DB2C)

4bytes: The DB2SUBSYSID indicates the location where the z/OS RMDB is located.

CM Library Server Database (NQADB2C)

Alias used to locate a given database within the specified DB2 subsystem.

Library Server Location (LOCAL/REMOTE)

This value indicates whether the Library Server database reside “locally” within the same DB2 subsystem as that of the z/OS RM, or resides externally.

Plan Name (OSAPFVTA)

8 bytes: The bind plan for the asynchronous replication process.

User ID (IFVTA)

: The Content Manager user with System Administrator authority.

User Password (IFVTA1)

The Content Manager System Administrator password.

Resource Manager Name (IMWEBSR1)

: This name indicates to which z/OS RM the asynchronous process is associated.

Sleep Value (300)

The duration of time in seconds for the process to pause before reprocessing items marked for deletion.

Tracelevel (0 – 3)

Log level. The current supported values are 0, 1, 2, and 3.

- a. At level 0: ERROR Logs only the Asynchronous Replicate Summary Report and errors
 - b. At level 1: INFO Logs the function entry/exit status plus ERROR.
 - c. At level 2: DETAIL Logs function input/output, success/failure results and INFO.
 - d. At level 3: DEBUG Logs verbose statements used for debugging and INFO.
2. If all the input parameters are valid, then the asynchronous replication process proceeds to connect to the Library server's database to retrieve the data-collection and resource manager (RM) information in the system to set up for the replication process. In addition, the readToBeReplicated function is called to fetch the information of the items to be replicated in the RM Asynch table and store them in the ReplicaItemStruct struct. This is the struct that drives the asynchronous replication process.
 3. Once the ReplicaItemStruct struct has been populated, asynchronous replication calls on the several helper functions to complete its tasks. The most important functions and their functionalities are the following:

mapRMData()

This function takes the ReplicaItemStruct struct as one of its parameters and use the information in it to validate the RM(s) listed in the Asynch table against the RM information retrieved from the LS to ensure that they exist. If the verification is successful, then the source and target RMs are mapped appropriately in the ReplicaItemStruct struct.

mapCollData()

This function also takes the ReplicaItemStruct struct as one of its parameters and use the information in it to validate the collection(s) listed in the Asynch table against the collection information retrieved from the LS to ensure that they exist. If the verification is successful, then the source and target collections are mapped appropriately in the ReplicaItemStruct struct. Note: this function is called only after the source and target RMs are mapped successfully.

checkoutItems()

This function also takes the ReplicaItemStruct struct as one of its parameters and use the information in it to check out the items to be replicated. When an item is checked out, not only is it locked but its information is also written into the CHECK OUT table in the LS.

sendReplicasToTargetRM()

This function takes the ReplicaItemStruct struct as one of its parameters and use the information in it to determine which item needs to be replicated and from which RM/collection to which RM/collection.

4. In greater detail, sendReplicasToTargetRM() accomplishes its tasks in several steps. First, it checks with OAM to determine if the object exists or not. If the object exists, then it attempts to retrieve the object. Second, it determines which target RM/collection this object needs to be sent to. Finally, an HTTP request with the target RM/collection information and the object is sent to the target RM/collection. Hence, the object is replicated. NOTE: asynchronous replication only happens as described above if asynchronous replication is enabled on beforehand. If asynchronous replication is not enabled and the user would like to replicate an object, the user must send an request with an IMPORTREPLICA order.

Table definitions and column descriptions

Table 16. ICMRMCONTROL

ICMRMCONTROL			
...			
STOPREPLICATOR	SMALLINT	NOT	NULL,
...			
STOPREPLICATOR: (e.g. (0/1)) The REPLICATOR bit determines if the ICMRMAP asynchronous replication process will stop processing entries after one round or not and is also the method by which the Asynchronous Replication process is stopped. If it's set to 1, then it will stop after 1 round of processing. If it's set to 0, then it depends on the sleep value to determine how long the process should pause before it starts to process again.			

Table 17. ICMRMASYNCH

ICMRMASYNCH	
ITEMID CHAR (26) NOT NULL,	
VERSIONID CHAR (5) NOT NULL,	
OBJID CHAR (44) NOT NULL,	
COLLNAME CHAR (44) NOT NULL,	
PRIMARY KEY (OBJID)	

ITEMID

Example: A1001001A03L09B64813I92553 The ItemId value is that generated by the Library Server (LS) during the object store transaction.

VERSIONID

Example: 1 In the case where versioning is enabled, this value denotes a given instance of an object.

OBJID

Example: A1001001.A03L09.B64813.I92553.V001 The ObjID is the itemid and versioned combined and "." delimited and represents a unique Content Manager (CM) object locator within Object Access Method (OAM).

COLLNAME

Example: CLLCT001 The sourcecollname is the collection under which the object was originally stored and is the source of the prefetch operation.

Table 18. ICMSTITEMSTODELETE

ICMSTITEMSTODELETE		
ITEMID	CHAR (26)	NOT NULL,
VERSIONID	SMALLINT	
RCODE	SMALLINT	

ITEMID

Example: A1001001A03L09B64813I92553 The ItemId value is that generated by the Library Server (LS) during the object store transaction.

VERSIONID:

Example: 1

In the case where versioning is enabled, this value denotes a given instance of an object.

RCODE

Example: 1

RM name, given by the user, is used to map to RCODE, which in turn points to an entry in the table.

Managing documents in DB2 Content Manager

DB2 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 DB2 Content Manager system. From the API level, the only difference between a document and a folder is the semantic type and the respective DKFolder functionality. A document is comprised of attributes, or metadata, that describe the document, including single valued attributes (document name, date, subject), multi-valued attributes (keywords), and collections of multi-valued attributes (address, consisting of street, city, state, and zip).

The document management data model uses document parts to associate objects (resource items) with the document. This model supports more than one part to construct a document. For example, each page could be a separate part. In order for an application to determine the order of the parts that make up a document, a part number is stored in the document parts. The document parts contains a pointer to the object (a reference attribute) which contains other information about the part such as MIME type, size, the resource manager ID which contains the part, the collection name on that resource manager and so forth. Every object can have different attributes. For example, an annotation might have X and Y coordinates, while a note log might have the CCSID of the text of the note.

To better understand the document management data model, consider the following scenario of a user who imports a document using a client application:

- A window is displayed to the user.
- The user enters (or selects) the name of the file to import into the system. For example, a police report.
- The user selects the type of document (memo, claim, design).
- A new window opens, where the user can enter attributes that describe the document. The date, a claim number, and insurance policy number, for example.
- The user defines a document parts and enters some values for the attributes. The police report is a document parts of a claim, for example.
- The user finishes entering the document descriptions and completes the task. The police report is created.

Using either the APIs or the JavaBeans, the client application then connects to the library server and the resource manager. The system creates two items (a non-resource item and a resource item) to store the document. Two items are

created because the police report contains a photograph, which is stored in the resource manager. The document is created with a single API call. The object is then stored in the resource manager and the resource manager returns the timestamp, and other metadata for the object. The resource manager creates a reference attribute for the object, and inserts the reference attribute into the child component of the document. A final call to the library server is made to store the child component and to update the attributes. The entire process is bound by a transaction so that any API failure does not result in a partially created document.

After you create a document, you can update it. You can perform two types of updates: change the metadata or change the content. The library server automatically creates a new item record with the next version number (if the item is version-enabled) and copies all of the child components associated with the item.

Creating the document management data model

This section helps you complete the main tasks associated with the document management data model:

- Creating a document item type.
- Create a document.
- Updating a document.
- Retrieving and deleting a document.
- Versioning of parts in the document management data model.

Creating a document item type

Java:

1. Create a document ItemType with ItemType classification = DK_ICM_ITEMTYPE_CLASS_DOC_MODEL, set the following.


```
docItemTypeDef.setVersionControl
    ((short)DKConstantICM.DK_ICM_VERSION_CONTROL_ALWAYS);
docItemTypeDef.setVersioningType
    (DKConstantICM.DK_ICM_ITEM_VERSIONING_FULL);
docItemTypeDef.setDeleteRule(DKConstantICM.DK_ICM_DELETE_RULE_CASCADE);
```
2. Create ItemType relation to add Parts to document. Retrieve EntityDef for each Part.


```
Parttype = (DKItemTypeDefICM) dsDef.retrieveEntity(PartName);
```
3. For each part, create a ItemType Relation and set the values.


```
DKItemTypeRelationDefICM itRel = new DKItemTypeRelationDefICM(ds);
itRel.setTargetItemTypeID(PartName);
itRel.setDefaultRMCode((short)1);
itRel.setDefaultACLCode(DKConstantICM.DK_ICM_SUPER_USER_ACL);
itRel.setDefaultCollCode((short)1);
itRel.setDefaultPrefetchCollCode((short)1);
itRel.setVersionControl(DK_ICM_VERSION_CONTROL_NERVER);
```
4. Add the ItemType relation to the Document (Source).


```
docItemTypeDef.addItemTypeRelation(itRel);
```
5. Add the document ItemType to persistent store.


```
docItemTypeDef.add();
```

C++:

1. Retrieve the content server definition object.


```
DKDatastoreDefICM * dsDefICM = (DKDatastoreDefICM *)dsICM->datastoreDef();
```

2. Retrieve the content server administration object.

```
DKDatastoreAdminICM * pdsAdmin =  
    (DKDatastoreAdminICM *)dsDefICM->datastoreAdmin();  
DKItemTypeDefICM *itemType = NULL;  
DKItemTypeRelationDefICM *itemTypeRel = NULL;  
DKAttrDefICM* attr = NULL;
```

3. Retrieve an attribute for the document item type. If the attribute does not exist, create it.

```
dkAttrDef *pAttr = dsDefICM->retrieveAttr("docTitle1");  
if(pAttr == NULL)  
{  
    attr = new DKAttrDefICM(dsICM);  
    attr->setName("docTitle1"); //attribute name column name  
    attr->setType(DK_CM_CHAR);  
    attr->setSize(100);  
    attr->setNullable(false);  
    attr->setUnique(false);  
    attr->add();  
    pAttr = attr;  
}  
itemType = new DKItemTypeDefICM(dsICM);  
itemType->setName("DocModelTest");  
itemType->setDescription("This is a test Item Type");  
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);  
itemType->setAutoLinkEnable(false);  
itemType->setVersionControl((short)DK_ICM_VERSION_CONTROL_ALWAYS);  
itemType->setVersioningType(DK_ICM_ITEM_VERSIONING_FULL);  
itemType->addAttr(pAttr);
```

4. Create a relation between the document that you just created and part1. To do that, first retrieve EntityDef for each Part.

```
DKItemTypeDefICM *itemTypePart1 = (DKItemTypeDefICM *)  
    dsDefICM->retrieveEntity("ICMBASE");  
//int part1ITypeid = itemTypePart1->getItemTypeId();  
int part1ITypeid = itemTypePart1->getIntId();
```

5. For each part, create an item type relation and set the values.

```
DKItemTypeRelationDefICM *itemTypeRelPart1=  
    new DKItemTypeRelationDefICM(dsICM);  
itemTypeRelPart1->setTargetItemTypeID(part1ITypeid);  
//sets the default resource manager  
itemTypeRelPart1->setDefaultRMCode((short)1);  
//sets the default ACL code  
itemTypeRelPart1->setDefaultACLCode(1);
```

6. Set the default collection where item resources pertaining to this item type are to be stored.

```
itemTypeRelPart1->setDefaultCollCode((short)1);
```

7. Set the default prefetch collection where item resources pertaining to this item type are to be stored.

```
itemTypeRelPart1->setDefaultPrefetchCollCode((short)1);  
itemTypeRelPart1->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);  
itemTypeRelPart1->setSourceItemTypeID(itemType->getIntId());
```

8. add the item type relation to the document (source).

```
itemType->addItemTypeRelation(itemTypeRelPart1);
```

9. Create a relation between the document that you just created and part2. To do that, first retrieve EntityDef for each part.

```
DKItemTypeDefICM *itemTypePart2 = (DKItemTypeDefICM *)  
    dsDefICM->retrieveEntity("ICMANNOTATION");  
int part2ITypeid = itemTypePart2->getIntId();
```

10. For each part, create an item type relation and set values.

```

DKItemTypeRelationDefICM * itemTypeRelPart2= new
    DKItemTypeRelationDefICM(dsICM);
itemTypeRelPart2->setTargetItemTypeID(part2ITypeId);
itemTypeRelPart2->setDefaultRMCode((short)1);
itemTypeRelPart2->setDefaultACLCode(1);
itemTypeRelPart2->setDefaultCollCode((short)1);
itemTypeRelPart2->setDefaultPrefetchCollCode((short)1);

itemTypeRelPart2->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);

itemTypeRelPart2->setSourceItemTypeID(itemType->getIntId());

```

11. Add the item type relation to the document (source).

```
itemType->addItemTypeRelation(itemTypeRelPart2);
```

12. Update the definition of the item type in the library server.

```
itemType->add();
```

For additional information, see the `SItemTypeCreationICM` sample.

Creating a document

An item with the "Document" semantic type can contain attributes (like items of other semantic types) and multiple "parts" (unlike items of other semantic types) inside it. The steps below take you through the process of creating an item (based on a pre-defined document item type) that contains one attribute and one "part". Note that in the steps below, it is assumed that an item type called "s_simple," with one attribute, called "S_varchar," and one part ("ICMBASE") has already been defined.

Java

1. Create the document DDO.

```
DKDDO ddoDocument = dsICM.createDDO("S_simple",
    DKConstant.DK_CM_DOCUMENT);
short dataId = 0;
String attrValue = "Test";
```

2. Set the document's attribute. In this case, we assume that the item type has only one attribute.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR, "S_varchar");
ddoDocument.setData(dataId, attrValue);
DKParts parts = null;
// Document's parts
```

3. Access the document's parts collection.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
    DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
    dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
        DKConstantICM.DK_CM_DKPARTS);
    parts = new DKParts();
    ddoDocument.setData(dataId, parts);
}
else
{
    parts = (DKParts) ddoDocument.getData(dataId);
    if (parts == null)
    {
        parts = new DKParts();
        ddoDocument.setData(dataId, parts);
    }
}
```

4. Create a part of pre-defined type "ICMBASE". This part will be added to the created document. It is assumed that the document created below is based on an item type with only one part.

```
DKLobICM pLobPart = (DKLobICM) dsICM.createDDO("ICMBASE",
    DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setPartNumber(1);
// Set the mime type for added part
pLobPart.setMimeType("text/plain");
String partValue = "This is a base part";
pLobPart.setContent(partValue.getBytes());
```

5. Add the created part to the "parts" collection. Note that this is a deferred save (the change is not committed to the datastore until the document DDO is persisted).

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. Persist document to the datastore.

```
ddoDocument.add();
```

C++

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a new DDO of type DocModelTest and semantic type DK_CM_DOCUMENT
DKDDO* ddoDocument = dsICM->createDDO("DocModelTest",DK_CM_DOCUMENT);
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("docTitle1")),DKString("this is a string value"));
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
    pdsDef->retrieveEntity("DocModelTest");
// Retrieve the collection of DKItemTypeRelationDefICM object for given source
// item type from the persistent store
dkCollection* pRelationColl = itemType->retrieveItemTypeRelations();
dkIterator* pIter = pRelationColl->createIterator();
int noOfPartsToCreate = pRelationColl->cardinality();
DKParts* pPartColl= NULL;
// Create the parts collection for the object if it does not exist
short dataId = ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
if (dataId ==0) {
    dataId = ddoDocument->addData(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
    pPartColl = new DKParts();
    ddoDocument->setData(dataId,pPartColl);
}
else {
    pPartColl =(DKParts*) (ddoDocument->getData(dataId).value());
    if (pPartColl ==NULL) {
        pPartColl = new DKParts();
        ddoDocument->setData(dataId,pPartColl);
    }
}
int i=0;
DKItemTypeRelationDefICM* itemTypeRelPart =NULL;
DKItemTypeDefICM* pEnt =NULL;
// CV v8 BLOB
DKLobICM* pPart =NULL;
DKString str = "This is to test the document model with two parts";
while(pIter->more()) {
    i=i+1;
    itemTypeRelPart=(DKItemTypeRelationDefICM*) pIter->next()->value();
    pEnt =(DKItemTypeDefICM*)
        ((DKDatastoreDefICM*) pdsDef)->retrieveEntity(
            (long)itemTypeRelPart->getTargetItemTypeID());
    pPart =(DKLobICM*) dsICM->createDDO(pEnt->getName(), DK_CM_RESOURCE);
    pPart->setPartNumber(i);
    pPart->setContent(str);

    DKAny any = (dkDataObjectBase*) pPart;
    pPartColl->addElement(any);
}
//Add the DDO to the datastore
ddoDocument->add();
```

For more information, see the SDocModelItemICM sample.

Updating a document

The steps below take you through the process of updating an item of semantic type "Document." In the steps below, a new part is added and the attribute value is updated.

Java

1. Update the attribute value for the document item.

```
String attrValue = "New Value";
short dataId=ddoDocument.dataId
(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
```

2. Access the document's parts collection.

```
DKParts parts = null;
// Document's parts
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
    dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
DKConstantICM.DK_CM_DKPARTS);
    parts = new DKParts();
    ddoDocument.setData(dataId, parts);
}
else
{
    parts = (DKParts)ddoDocument.getData(dataId);
    if (parts == null)
    {
        parts = new DKParts();
        ddoDocument.setData(dataId, parts);
    }
}
```

3. Create data for the new part.

```
String partValue = "This is an annotation";
```

4. Create a part of pre-defined type "ICMANNOTATION". This part will be added to the created document. Here, it is assumed that the document being created is based on an item type with only one part. Once the new part is added, the document will have two parts.

```
DKLobICM pLobPart = (DKLobICM)dsICM.createDDO("ICMANNOTATION",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setContent(partValue.getBytes());
pLobPart.setPartNumber(2);
```

5. Add the created part to the "parts" collection. Note that this is a deferred save (the change is not committed to the datastore till the document DDO is persisted).

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. Persist the changed document to the datastore.

```
ddoDocument.update();
```


C++

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a Document DDO from a PID string
DKString pidString = ....;
//...
DKDDO* ddoDocument = dsICM->createDDO(pidString);
DKPidICM* pPID = (DKPidICM*) ddoDocument->getPidObject();
DKString* pstrItemType = &pPID->getObjectType();
// Retrieves the definition for the item type "DocModelTest" from the
// persistent datastore. The definition is returned as a the *dkEntityDef
// object that is in turn typecast to a DKItemTypeDefICM object
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
    pdsDef->retrieveEntity(*pstrItemType);
ddoDocument->setData(ddoDocument->dataId(
    DK_CM_NAMESPACE_ATTR,DKString("docTitle1")),
    DKString("this is a new string value"));

DKString updateString = "This is updated part";
DKSequentialIterator* pSeqIter = NULL;
DKLobICM* pPart = NULL;
short dataId = ddoDocument->dataId (DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS);
DKParts* pParts = (DKParts*) &ddoDocument->getData(dataId);
if (pParts != NULL) {
    pSeqIter = (DKSequentialIterator*) pParts->createIterator();
}
else return; // quit
pPart = (DKLobICM *) pSeqIter->next();
// Update the existing part with the new content
pPart->setContent(updateString);
// Add a new part to the Document
DKLobICM* pPart1 =
    (DKLobICM*) dsICM->createDDO ("ICMNOTELOG", DK_CM_RESOURCE);
pPart1->setPartNumber(3);
DKString pTempData = "This is to test the document model with two parts";
pPart1->setContent(pTempData);
DKAny any = (dkDataObjectBase*)(DKDDO*) pPart1;
pParts->addElement(any);
//Update the DDO information in the datastore.
ddoDocument->update();
delete pSeqIter;
```

For additional information, see the SDocModelItemICM sample.

Retrieving and deleting a document

To retrieve a document, call `ddo.retrieve(option)`. If you set option to `DK_ICM_CONTENT_YES`, the parts TOC list, as well as the parts, is retrieved. Otherwise, only the TOC list of the parts is retrieved.

To delete a document, call `ddo.del()`. The document and its attached parts is deleted. For more information about retrieving and deleting documents with parts, see the SDocModelItemICM API sample.

Versioning of parts in the document management data model

Versioning properties of document parts are determined by the versioning properties of the item type relation to each part type selected in the document's item type. Versioning characteristics of document parts include the following:

- Like regular documents, parts can have one of three versioning models: versioned-always, versioned-never (default) and application-controlled versioning.
- If an item type has a version policy of versioned-never, its parts also have a versioning policy of versioned-never.
- If an item type T has a version-policy of versioned-always and an item I of item type T and you modify (by adding/deleting or updating a part) either its attributes or its parts collection, a new version of item I is created.
- Parts of documents, unlike documents themselves, do not have a maximum number of versions.
- You can obtain part-level versioning rules from the item type relation object for the part of interest (base, note, annotation etc.).

The examples below show how to get the versioning rules for an item type's base part.

Java

```
String itemTypeName="book"; //example item type
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM item =null;
DKDatastoreICM ds = new DKDatastoreICM();
...
item = (DKItemTypeDefICM)ds.datastoreDef.retrieveEntity(itemTypeName;
DKItemTypeRelationDefICM itemRelation =
(DKItemTypeRelationDefICM)item.retrieveItemTypeRelation(partId);
versionControlPolicy = itemTypeRelation.getVersionControl();
```

C++

```
DKString itemTypeName="book"; //example item type
//Specify the part id for which we need versioning information
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM * itemType;
//Retrieve the entity corresponding to the "book" item type
itemType =
(DKItemTypeDefICM *)dsICM->datastoreDef()->retrieveEntity(itemTypeName);
//Retrieve the relation object for the specified part id
DKItemTypeRelationDefICM * itemTypeRelation =
(DKItemTypeRelationDefICM*)itemType->retrieveItemTypeRelation(partId);
//Retrieve the version control policy for the specified part
int versionControlPolicy = itemTypeRelation->getVersionControl();
```

Working with transactions

Transactions allow DB2 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 (transaction), but using this type of transaction can introduce some performance overhead.

When a transaction ends, the entire transaction is either committed or rolled back. If it is committed, all of the Content Manager server changes made by API calls within the transaction are permanent. If a transaction is rolled back or fails, all the changes made within the transaction are reversed during rollback processing.

The commit and rollback of a transaction are done automatically in the case of implicit transactions. The Content Manager system initiates a rollback when a severe error occurs or when it is necessary to resolve a deadlock between the library server and the database.

Within a transaction, uncommitted resource manager changes are not visible to the application that made the changes until the transaction is committed. For example, you make changes to a resource manager item and you store it. If you retrieve that item before the transaction is committed, the item will not reflect the changes that you just made. You will not see the updated item until the transaction is committed.

Concurrent or overlapping transactions through a single library server connection are not supported. To maintain concurrent transactions, you must make multiple connections between the library server and the database, or initiate multiple client processes or threads if you are working with a client application. Applications like IBM WebSphere[®] Application Server handle processes, connections, and sessions.

The `execute()` and `executeWithCallback()` methods in `DKDatastoreICM` automatically create an additional connection to the database when invoked. The new database connection is then used to execute the query. Since queries use a separate database connection, they also have a separate transaction scope from other content server operations. The connection to the database is closed (or returned to the pool, if pooling is enabled) when the `DKResultSetCursor` is closed.

Things to consider when designing transactions in your application

If a client node or library server fails before the transaction is committed, the database recovery function rolls back the transaction on the library server immediately. The resource manager changes made during the failure are undone immediately if the client node and resource manager are both active. If the client node itself failed, you should put the resource manager through a cycle of the Asynchronous Recovery Utility in order to restore consistency between the resource manager and the library server. Before the utility runs, the servers still have data integrity. What is affected are operations on the in-progress items that had the failure, which will be rejected until the RM is recovered. Failure during in-progress update of an object prevents another update of that same object, until the first failure is reconciled.

If the resource manager fails, you should run the asynchronous recovery utility to remove inconsistencies. On OS/390, the resource manager has native transaction capabilities, such as Object Access Method (OAM), which are used to recover more expediently.

Caution when using transactions

For 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 of 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) 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 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 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 Application Programming Reference for details. Some administrative APIs cannot be included in 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 transaction.

DKDatastoreICM.commit()

Commits transaction changes to the database.

DKDatastoreICM.rollback()

Rolls back or removes transaction changes from the database.

DKDatastoreICM.checkOut()

Acquires a persistent write lock on an item.

DKDatastoreICM.checkIn()

Releases a previously acquired persistent write lock.

DKDatastoreICM.add()

Creates a new item in the database.

DKDatastoreICM.updateObject()

Updates an item. The item must be checked out prior to calling this method.

DKDatastoreICM.retrieveObject()

Retrieves an item from the database.

DKDatastoreICM.deleteObject()

Deletes an item from the database.

DKDatastoreICM.moveObject()

Re-index an item. Moves an item from one item type to another item type. The item must be checked out prior to calling this method.

Another source for information about transactions is the SItemUpdateICM sample.

New explicit transactions behavior in Version 8.3

If the DB2 Content Manager system detects an error during a persistent operation, the system's default behavior is to always automatically roll back an explicit transaction. For example, XYZ Insurance has an online policy payment system, and a user submitted their credit card information for processing, but the power suddenly goes out. The credit card charge and the payment update to the policy are undone by the system.

Automatic rollback of explicit transactions is important because you cannot rely on your application to reverse a series of updates that it made prior to a failed operation, especially for important operations. Perhaps the connection is lost or your system crashed due to a power failure and you cannot undo the prior transactions. Your application might not remember what it was doing before the crash if it did not have a logging and recover mechanism. With a transaction, you tell the system to start recording your changes, but drop them if you do not explicitly tell it that you are done.

If you want to override the system's default behavior for explicit transactions, see the STransactionsRollbackChangeICM sample.

Transaction behavior when deleting a user does not belong in a group

By default, if you attempt to delete a user from a group, and that user does not exist (or belong) in that group, the `DKUserMgmt::update` method rolls back the transaction. If you do not want the default behavior in your application, and you want to be able to delete a user, even if they do not belong in the group, you can use a new method. You can find the new method in `DKUserMgmt`. Following is the method signature:

```
public void update(dkUserGroupDef userGroup, Vector v, int action, int option)
```

The constant `DK_ICM_DO_NOT_ROLLBACK_ON_ERROR`, from `DKConstantICM.java`, allows you to override the default behavior. If the value for the action argument is `ACTION_DELETE` and you pass zero as the value for the option argument, this method behaves the same as the current update method, which has the following signature:

```
public void update(dkUserGroupDef userGroup, Vector v, int action)
```

If the value for the action argument is `ACTION_DELETE`, and the value for the option argument is `DK_ICM_DO_NOT_ROLLBACK_ON_ERROR`, the new update method does not roll back the transaction, even if the user being deleted from the group does not belong to the group. Again, the default behavior (if user does not belong to the group) is to throw a `DKNotExistException` exception.

For more information see the `DKUserMgmt` API in the Application Programming Reference.

Chapter 5. Searching for data

When an application searches for items stored in the DB2 Content Manager system, the underlying engine processes the search based on a query that you compose. To efficiently traverse DB2 Content Manager's hierarchical data model, you compose queries using the DB2 Content Manager query language, a powerful XML-based query language. The query language provides the following benefits:

- Supports the full data model.
- Supports searching for a specific version, such as the most current version.
- Enables searches within component type view hierarchies across linked items, and references.
- Combines parametric and text search.
- Provides sorting capabilities, using SORTBY.
- Enforces DB2 Content Manager access control.
- Conforms to XQuery Path Expressions (XQPE) standards, a subset of the W3C XML Query working draft.
- Completes 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 the W3C XML Query working draft. Using the query language, you can search hierarchical item types and locate items quickly and easily. To begin writing 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). Note that in the system administration client, component type views are called component type subsets.

When you submit a query for a document, folder, object, and so forth, your request is directed to the DB2 Content Manager query engine. The engine processes the query and translates it to the appropriate SQL. The library server then adds in the security checks (ACLs) and runs the query.

Important: Query and search are used interchangeably in this section, and have essentially the same meaning.

Querying the DB2 Content Manager server

When you query the DB2 Content Manager library server, you complete the following main steps:

1. Create a query string to represent your search conditions.
2. Call the `evaluate`, `execute`, or `executeWithCallback` method with your query string and query options to get the results or call the `executeCount` method to get the count of the results.
3. Receive the results through a `DKResults` object, a `dkResultSetCursor`, or a `dkCallback` object.

The query APIs perform the query processing tasks, such as preparing and executing a query, monitoring the status of a query execution, and retrieving the results.

You can build queries that perform three main types of searches: parametric, text, and combined parametric and text. A parametric query uses conditions such as equality and comparison. A text query uses text search functions, which makes the search more thorough. A combined query is composed of both text and parametric conditions.

To run a query, you can use one of the following methods: `evaluate`, `execute`, or `executeWithCallback`. The `evaluate` method returns all results as a collection of DDO's. With the default options, this works well for result sets that are relatively small, 200 items or less. When working with larger results sets, you can specify the `IDONLY` retrieve option to prevent attribute data from being stored in the DDOs that are returned. In this case, you must explicitly retrieve the attribute data for the DDOs as the data is needed by the application. Also note that you can limit the result set size using the *max results* option. The `execute` method returns a `dkResultSetCursor` object, which has the following characteristics:

- The `dkResultSetCursor` works like a content server cursor.
- You can use it for large result sets because the DDOs it returns are retrieved in blocks as the user requests them.
- You can use `dkResultSetCursor` to rerun a query, by calling the `close` and `open` methods.
- You can use the `dkResultSetCursor` to delete and update the current position of the result set cursor.
- The `dkResultSetCursor` holds a database cursor open to fetch rows as they are needed from the database.
- To prevent locking and transaction problems, a new database connection is used to execute the query. This means that your application might use two connections while the cursor remains open. To avoid this additional connection, use the `evaluate` method to run the query.

The `executeWithCallback` method executes a query in an asynchronous manner and sends the results to the specified callback object in blocks. As such, the `executeWithCallback` method frees up the main thread of execution from the task of retrieving query results. For more information on query methods, see the `SSearchICM` sample. See the Application Programming Reference for the `evaluate`, `execute`, and `executeWithCallback` methods in `DKDatastoreICM` class.

Starting with DB2 Content Manager Version 8 Release 3, you can get the count of query results without getting the results themselves. There are two ways that you can do this. One is through the `executeCount` method in `DKDatastoreICM`, and the other is through the `cardinality` method in `dkResultSetCursor`. The `executeCount` method in `DKDatastoreICM` takes in a query string and options and returns the estimated count of the results of the query. The `cardinality` method in `dkResultSetCursor`, implemented in the DB2 Content Manager Version 8 connector, allows you to obtain the count when you already have a `dkResultSetCursor`. This call requires a separate trip to the server to evaluate the count.

Note that the count of query results can be different from the number of results that you would get when running the query and getting the results. This is because the count is only accurate at an instant in time. Items might be added or deleted between the time the count is retrieved and when the results are retrieved.

There are also a few cases in which query might return the ID of an item, but the retrieve API cannot access that item. In these cases the count will also be inaccurate. Also note that to get the count, a server call must be made to evaluate the query. This means that if you want the count and then the results, you must make two trips to the server.

Applying the query language to the DB2 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, DB2 Content Manager data model elements are represented as follows:

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

Root components

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

Root components can contain an ICMCHECKEDOUT element to indicate an item's "checked out" status. It has two attributes: ICMCHKOUTUSER to indicate which user checked out the item, and ICMCHKOUTTS, which is the timestamp for when the item was checked out, for example 2003-08-02-17.29.23.977001).

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 DB2 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, 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. You can use links to model a many-to-many relationship between items. Note that this relationship is only between root components (a link cannot originate or end in a child component).

The links originating at an item are represented by <OUTBOUNDLINK> XML elements with the following attributes: IDREF LINKITEMREF, IDREF TARGETITEMREF and STRING LINKTYPE. The LINKITEMREF is a reference to an item that contains meta-data for the link. The TARGETITEMREF is a reference to the item pointed to by the link. The

LINKTYPE is the type of the link. Similarly, links pointing to an item are represented by <INBOUNDLINK> XML elements with the following attributes: IDREF LINKITEMREF, IDREF SOURCEITEMREF and STRING LINKTYPE. The SOURCEITEMREF is a reference to the item where the link originates. For more information on link semantics, see the SLinksICM and SSearchICM samples.

References (reference attributes)

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

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

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

References also support delete semantics (restrict, cascade, set null, or no action). For more information on reference attributes, see the SReferenceAttrDefCreationICM and SSearchICM samples.

Documents

Folder functionality provided in Content Manager through the DKFolder is stored as a set of links of the "DKFolder" (DK_ICM_LINKTYPENAME_DKFOLDER) link type. Each folder-content relationship is modeled as an outbound link from the folder and an inbound link to the contents making the folder the source and the contents the targets of each link. Follow the semantics of searching and traversing links in order to search and traverse folders. For more information, see the SFolderICM and SSearchICM samples.

Understanding parametric search

Items are often retrieved by initiating a search on selected attributes. A single query can examine both system-defined and user-defined attributes of the items in the content server. Simple search conditions consist of an attribute name, an operator, and a value that are combined into a clause. DB2 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"
"IN"
"NOT IN"

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

Understanding text search

The DB2 Content Manager Version 8 Release 3 supports two types of text search: text search of attributes that contain text in components and text search of objects. The main difference between the two types of text search is how the content is stored. When you define an attribute to be text searchable, you are indicating that one can search text contained in the column of that attribute.

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".

To make an attribute (column) text searchable, you must create text indexes on item type content or specific attributes using either the system administration client or system administration APIs.

A text index holds information about the text that is to be searched. This information is used to perform text search efficiently. Version 8 Release 3 now uses the following class hierarchy to represent these text indexes:

dkTextIndexICM

Class interface that represents a text index.

dkDB2TextIndexICM

Class interface that extends `dkTextIndexICM`, and represents the text index for IBM DB2 Universal Database Net Search Extender. This was formerly known as IBM DB2 Universal Database Text Information Extender in DB2 Content Manager Version 8.1. In the discussion of text search in the ICM query language, DB2 Net Search Extender and DB2 Text Information Extender are used interchangeably. From the perspective of the ICM query language, DB2 Net Search Extender and DB2 Text Information Extender do not differ in text search syntax or functionality.

For information about DB2 Text Information Extender, see *IBM DB2 Universal Database: Text Information Extender Administration and User's Guide*. For information about DB2 Net Search Extender used in DB2 Content Manager Version 8.2 or higher, see *IBM DB2 Universal Database: Net Search Extender Administration and User's Guide* in the DB2 UDB Information Center.

dkOracleTextIndexICM

Class interface that extends `dkTextIndexICM`, and represents an OracleText index.

| Note that Version 8 Release 3 still supports Version 8 functionality such as the
| definition class DKTextIndexDefICM and the methods using DKTextIndexDefICM
| of DKItemTypeDefICM and DKAttrDefICM. However, the old functionality is not
| supported with OracleText.

Searching for object contents

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

Searching for documents

You can perform text search on the contents of document parts. A virtual component type view "ICMPARTS" is supported in query as a child of every document in the system. The "TIEREF" attribute under the "ICMPARTS" component type view refers to the contents of all the text-searchable parts of that document for text search purposes. See the 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 dkDB2TextIndexICM or dkOracleTextIndexICM interfaces. For more information on the APIs, see the Application Programming Reference or the SItemTypeCreationICM sample.

Understanding text search syntax

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

Basic text search

Since the majority of text searches are done by simply listing a few words one after the other, basic (simplified) text search syntax was designed specifically to make this most common case easy for users. The syntax also allows for use of "+" and "-", as well as for use of quoted phrases. Basic search is supported on both IBM DB2 UDB and Oracle library servers.

Simplified text search is done by using the advanced text search 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. Remember 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. See the query examples for information about how to use these functions.

Additional information about basic text search syntax includes the following:

- You must use advanced search to perform case-sensitive search.
- Terms within quotes are assumed to be a phrase.

- Use of + (plus) - (minus)
 - + (plus) = document must include this word.
 - - (minus) = document must not include this word.
 - When a + or - is not specified, the query engine uses an algorithm to match the words to the text.
- Boolean operators (AND, OR, NOT) are not valid and are ignored.
- Parentheses in the basic syntax are not supported.
- Valid wildcards
 - ? (question mark) = represents a single character
 - * (asterisk) = represents any number of arbitrary characters

For more information on basic text search, see the SSearchICM sample.

Advanced text search

Advanced text search syntax is used to allow the user to specify more complex conditions for text search. It allows such powerful features as proximity search and fuzzy search.

Advanced text search syntax uses database-specific functions similar to the way the "contains-text-basic" and "score-basic" functions are used for basic text search. Advanced search uses the search syntax supported by the underlying text engine (DB2 Net Search Extender for DB2 UDB and OracleText for Oracle), so the search syntax will be different.

contains-text-db2 and score-db2

Uses NSE text search syntax with one exception: you must change double quotes to single quotes, and vice-versa. For example, the CONTAINS (description, ' "IBM" ') = 1 condition in NSE would become contains-text-db2(@description, " 'IBM' ") = 1 in the DB2 Content Manager query language. This needs to be done to support simplicity of writing queries with minimal use of escape characters. Remember to equate the "contains-text-db2" function to 1 to check if it is true, and to equate it to 0 to check if it is false. See the query examples for more details on advanced text search.

Example:

```
/Book[contains-text-db2(@Title, "ibm")=1] SORTBY(score-db2(@Title, "ibm"))
```

For backward compatibility with DB2 Content Manager Version 8 Release 2, the advanced text search functions "contains-text" and "score" are still supported. These functions are identical to "contains-text-db2" and "score-db2" functions.

contains-text-oracle and score-oracle

Uses OracleText search syntax. Performs the same quote translation as contains-text-db2. For more information about supported syntax see OracleText documentation.

Example:

```
/Book[contains-text-oracle(@Title, "ibm", 1) > 0] SORTBY(score-oracle(1))
```

The contains-text-oracle function contains the optional third parameter: label name, which is used as the parameter to score-oracle.

For more information on advanced text search, see the SSearchICM sample.

Creating combined parametric and text search

Searches can be performed on virtually any piece of an item or component, text within an item or component, or text within resource content. Search can be one of the following types:

Parametric search

Searching is based on item and component properties, attributes, references, links, folder contents, and so forth.

Text Search

Searching within text.

Combined Search

Searching using both parametric and text search.

Follow the steps below to create a combined parametric and text search.

Java

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

```
String queryString =  
    "//Journal_Article [Journal_Author/@LastName = \"Richardt\" +  
    \" AND contains-text (@Text, \" 'Java' & 'XML' \")=1]\";
```
3. If you want to use retrieve options that are different from the defaults, use the DKNVPair array to package the options.

```
DKNVPair parms[] = new DKNVPair[3];  
String strMax = "5";  
parms[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, strMax);  
parms[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,  
    DKConstant.DK_CM_CONTENT_ATTRONLY |  
    DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);  
parms[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
```
4. Execute the combined query in one of three different ways: evaluate, execute, and executeWithCallback.

```
DKResults resultsCollection =  
    (DKResults)dsICM.evaluate(queryString,  
    DKConstant.DK_CM_XQPE_QL_TYPE, parms);
```
5. Process the results. The procedure for handling the results depends on which execution method you used.

For a complete sample and additional documentation, see the SSearchICM ICM API education sample in IBMCMROOT/samples/java/icm.

C++

1. Specify the search options. Note that the options array always has to have an end element. For example, if you want to specify two options, the options array must have three elements.

```
DKNVPair * parms = new DKNVPair[3];
DKNVPair * pparm = NULL;
DKString strMax = "5";
DKAny * anyNull = new DKAny();
//Allow a maximum of 5 items to be returned from the search
pparm = new DKNVPair(DK_CM_PARAM_MAX_RESULTS, strMax);
parms[0] = *pparm;
delete pparm;
//Specify what content is to be retrieved
pparm = new DKNVPair((long)DK_CM_PARAM_RETRIEVE,
    DK_CM_CONTENT_ATTRONLY | DK_CM_CONTENT_LINKS_OUTBOUND);
parms[1] = *pparm;
delete pparm;
pparm = new DKNVPair(DK_CM_PARAM_END, *anyNull);
parms[2] = *pparm;
delete pparm;
```

2. Execute the search. There are three ways to execute a search:

evaluate

Returns all the results as a collection; good for small sets.

execute

Returns a result set cursor, which the caller uses to iterate over the results.

executeWithCallback

Creates a thread that iterates over the result set and calls the callback object for each block of results. Caller uses the callback object to get the results

In the example below, only five results are desired, so the `DKDatastoreICM.evaluate` method is used.

```
DKResults * resultsCollection = (DKResults *) (dkCollection *)
    dsICM->evaluate(queryString, DK_CM_XQPE_QL_TYPE, parms);
```

3. Display the results of the search.

```
// Create an iterator to go through Results collection.
dkIterator* iter = resultsCollection->createIterator();

cout << "Results:" << endl;
cout << "    - Total: " << results->cardinality() << endl;

while(iter->more())
{
    //Each element in the returned array is an item (DDO)
    DKDDO* ddo = (DKDDO*) iter->next()->value();
    cout << "    - Item ID: " << ((DKPidICM*) ddo->getPidObject())
        ->getItemId() << " (" << ((DKPidICM*) ddo->getPidObject())
        ->getObjectType() << ")" << endl;
}
```

4. Clean up.

```
delete(iter);
delete[] parms;
....
```


For a complete sample and additional documentation, see the SSearchICM ICM API education sample in `IBMCMROOT/samples/cpp/icm`.

Example searches using the query language

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

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

The sample queries in the following sections are based on the query examples data model outlined in Figure 11 on page 197.. See the data model illustration when you review the sample queries.

For additional query syntax and examples based on an alternate data model, see the SSearchICM sample.

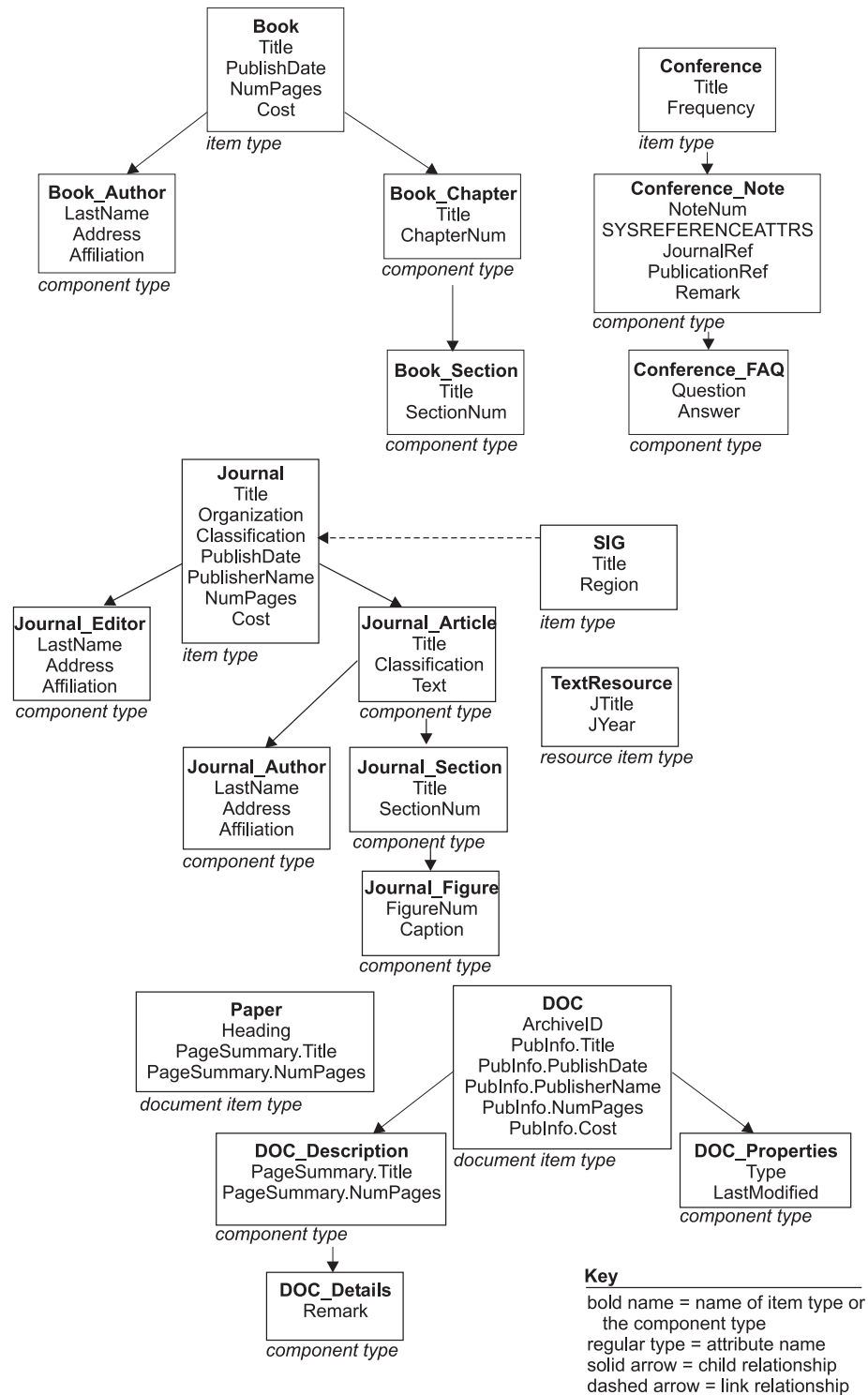


Figure 11. Query examples data model

The XML document below is a representation of the data model in figure Figure 11.

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>)
<ICMCKECKEDOUT (STRING ICMCKKOUTUSER, TIMESTAMP ICMCKKOUTTS)
</ICMCKECKEDOUT>
... (repeating <ICMCKECKEDOUT>)

</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>)

    <ICMCKECKEDOUT (STRING ICMCKKOUTUSER, TIMESTAMP ICMCKKOUTTS)
    </ICMCKECKEDOUT>
    ... (repeating <ICMCKECKEDOUT>)

</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 11 on page 197, and the sample XML document on page 197. Here are some hints to help you understand the query examples:

- Follow the query string as you would follow a directory structure
- "/" single slash indicates a direct child relationship
- "/" double slash indicates either a child relationship or a descendant relationship
- "." (DOT) represents the current component in the hierarchy
- ".." (DOT-DOT) represents the parent of the current component
- "@" (AT sign) denotes an attribute
- "[]" (square brackets) denote a conditional statement or a list

- "=>" (DEREFERENCE operator) represents linking or referencing action
- The result of the query must be a component (for example, an attribute cannot be the last thing in the path)

Additional examples and documentation are provided in the SSearchICM sample.

Example 1: access to components

This query finds all journals.

```
/Journal
```

Explanation:

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

Example 2: access to attributes

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

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

Explanation:

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

Example 3: multiple item types

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

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

OR

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

Explanation:

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

Example 4: arithmetic operations in conditions

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

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

Explanation:

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

Example 5: traversal of links in the forward direction

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

```

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

```

Explanation:

This is an example of following links in the forward direction. The virtual XML component OUTBOUNDLINK and its attribute TARGETITEMREF are used to traverse to all Journals and then finally the underlying Journal_Articles. The last component in the path is what is returned as the result of the query. The result can be constrained by traversing only specific link types ("contains" in this example) to a specific type of items (Journal in this example). Since the conceptual XML representation of the library server looks at inbound and outbound links as being parts of items, the dereferencing operator can be used to relieve applications from writing explicit joins.

Example 6: traversal of links in the backward direction

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

```

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

```

Explanation:

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

Example 7: advanced 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.

DB2

```

//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text-db2(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))

```

Oracle

```

//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text-oracle(@Text, " 'Java' & 'XML' ", 1)>0]
SORTBY(score(1))

```

Explanation:

This is an example of performing text search with the contains-text-db2 or contains-text-oracle functions. For the syntax supported by this function, see the *IBM DB2 Universal Database: Net Search Extender Administration and User's Guide* in the DB2 UDB Information Center. Note that the contains-text-db2 function should be equated with 1 to be true and 0 to be false. The score function uses the ranking information returned by NSE, which is used in this case to sort the resulting journal articles through SORTBY.

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

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

DB2

```
/Journal  
[Journal_Article[contains-text-db2(@Title, " 'Design' |  
  'Index' ")=1]]  
SORTBY (@Title DESCENDING)
```

Oracle

```
/Journal  
[Journal_Article[contains-text-oracle(@Title, " 'Design' |  
  'Index' ")>0]]  
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-text-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”.

DB2

```
/TextResource[contains-text-db2(@TIEREF, " 'Java' & 'XML'  
  ")=1]
```

Oracle

```
/TextResource[contains-text-oracle(@TIEREF, " 'Java' & 'XML'  
  ")>0]
```

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". DB2 Text Information Extender syntax is used as usual in this case inside the contains-text-db2 function. For the syntax supported by this function, see the *IBM DB2 Universal Database: Net Search Extender Administration and User's Guide* in the DB2 UDB Information Center.

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 Information Integrator for Content.

```
/Conference/Conference_Note [@PublicationRef =>
Book[@Title LIKE "%IIP%"]]
/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.

DB2

```
/Doc[contains-text-db2(.//ICMPARTS/@TIEREF, " 'XML' ")=1]
```

Oracle

```
/Doc[contains-text-oracle(.//ICMPARTS/@TIEREF, " 'XML' ")>0]
```

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 literals

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

```
/Book[@Cost IN (10, 20, 30)]
```

Although it is possible to perform the same query using the list operator, as in `/Book[@Cost = [10, 20, 30]]` (sub-optimal), for a large number of literals this approach might lead to errors because the generated SQL would be too long or too complex. If all the elements in the list are literals, always use the IN operator for the best performance and the shortest SQL. You can use the IN operator for literals of any type, including non-numeric types.

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`.

Example 28: checked out items

This query finds all items of the “Journal” item type that are currently checked out.

```
/Journal [ICMCHECKEDOUT]
```

Explanation:

The ICMCHECKEDOUT XML element is a sub-element of only the root components, but not of the descendant components. Therefore, if the ICMCHECKEDOUT element is written in a query as a condition of a child component (for example, `//Journal_Author [ICMCHECKEDOUT]`), then no results return.

Whenever an item is checked out, all versions of that item are checked out. Therefore, when an ICMCHECKEDOUT element is applied to a checked out item, all currently available versions will be returned. To retrieve a specific version, you can still use the `@VERSIONID` query syntax (for example, `/Journal [ICMCHECKEDOUT AND @VERSIONID = 4]`). For the latest version, you can use the `latest-version()` function.

Example 29: checked out items by person

This query finds all items checked out by “SMITH”:

```
/Journal [ICMCHECKEDOUT/@ICMCHKOUTUSER = "SMITH"]
```

Explanation:

The value for ICMCHKOUTUSER must be entered in upper case in a query. Since the content servers store userIDs as uppercase, all queries must query for userIDs using upper case. All attribute data pertaining to userIDs must store them in upper case as well.

Example 30: checked out items by timestamp

This query finds all items checked out after "2003-08-02-17.29.23.977001"
/Journal [ICMCHECKEDOUT/@ICMCHKOUTTS > "2003-08-02-17.29.23.977001"]

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 (+).

Using escape sequences in your queries

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 backslash). 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

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

Double quotation mark "

Precede your double quote with another double quote.

Example:**DB2**

```
//Journal_Article[contains-text-db2 (@Title, " 'Analysis of
""The Time Machine"" '%' ")=1]
```

Oracle

```
//Journal_Article[contains-text-oracle (@Title, " 'Analysis of
""The Time Machine"" '%' ")>0]
```

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:

DB2

```
/Book[contains-text-db2 (@Title, " 'Uncle Tom''s Cabin' ")=1]
  SORTBY (score-db2 (@Title, " 'Uncle Tom''s Cabin' "))
```

Oracle

```
/Book[contains-text-oracle (@Title, " 'Uncle Tom''s Cabin'
")>0]          SORTBY (score-oracle (@Title, " 'Uncle Tom''s
Cabin' "))
```

Note that Tom''s has two apostrophes.

Example B:**DB2**

```
/Book[contains-text-db2 (@Title, " ('Greek' & 'Plato''s
Symposium') & NOT ' Socrates' ")=1] SORTBY (score-db2 (@Title,
" ('Greek' & 'Plato''s Symposium') & NOT ' Socrates' "))
```

Oracle

```
/Book[contains-text-oracle (@Title, " ('Greek' & 'Plato''s
Symposium') & NOT ' Socrates' ")=1] SORTBY (score-oracle
(@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-db2 (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' ")=1]      SORTBY (score-db2 (@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-db2 (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' | 'Yahoo! For Dummies' | 'Usage of underscore !_ on
Yahoo!!' ESCAPE '!' | 'War and Peace' ")=1]
```

Note that an ESCAPE clause must be added after every term in your text search string where you escape wildcards, even if the escape character is the same in all the terms.

Using escape sequences with basic text search (contains-text-basic and score-basic functions)

Double quotation mark "

Precede your double quote with another double quote.

Example:

```
//Journal_Article[contains-text-basic (@Title, "Analysis of ""The  
Time Machine"" ")=1]
```

Since the article's title contains the name of the book in double quotes, "The Time Machine", these internal double quotes need to be escaped. The book title is inclosed in apostrophes to keep it as a phrase.

Single quotation mark (apostrophe) '

Precede the apostrophe with another apostrophe. Basic text search syntax allows terms enclosed within single quotes, so that a term can contain a space. The doubling of the apostrophe is therefore necessary to differentiate the case of an apostrophe occurring within a term from the case of an apostrophe starting a new term.

Example A:

```
/Book[contains-text-basic (@Title, "Uncle Tom's Cabin")=1]  
SORTBY (score-basic (@Title, "Uncle Tom's Cabin"))
```

Note that Tom's has two apostrophes.

Example B:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato's Symposium'  
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek  
+'Plato's Symposium' -Socrates "))
```

Note that Plato's has two apostrophes and 'Plato's Symposium' is enclosed in single quotes since it is a phrase.

Wildcards ("*", "?", and "\")

Precede "*", "?", and "\" characters with a backslash "\" if these characters are not to be treated as wildcards.

Star "*" is a wildcard character used to represent any number of arbitrary characters in basic text search for the functions contains-text-basic and score-basic. The question mark "?" is a wildcard character used to represent a single arbitrary character. For basic text search, the query language provides an escape character backslash "\" to be used when the term to be searched contains the wildcard character in it and you want to treat that wildcard character as a regular character.

Example A:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato*s*S?mposium'  
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek  
+'Plato*s*S?mposium' -Socrates "))
```

This example shows how to use basic text search when the spelling of a term is not certain. The "*" and "?" characters are meant to be wildcards in this case, so they are not escaped.

Example B:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1]  
SORTBY (score-basic (@Title, "Why forgive\?"))
```

In this example, the title contains the question mark "?" as a normal character, so this character can be escaped with a backslash.

Example C:

```
//Journal_Section[contains-text-basic (@Title,  
"C:\\OurWork\\IsNeverDone")=1]          SORTBY (score-basic (@Title,  
"C:\\OurWork\\IsNeverDone"))
```

Each backslash that naturally occurs in the search term "C:\\OurWork\\IsNeverDone" must be escaped with another backslash.

Using escape sequences in Java and C++

Precede special characters (for example, double quotes and backslash) with a backslash.

Example:

Query:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1]
```

Java

```
String query = "/Book[contains-text-basic (@Title, \"Why forgive\\?\")=1]";
```

C++

```
DKString query ("/Book[contains-text-basic (@Title, \"Why forgive\\?\")=1]");
```

Note how the internal double quotes and the backslash before the question mark are preceded by a backslash. This handling is inherent to Java and C++ programming languages. For more information, see the specifications for these languages.

Understanding row-based view filtering in query

With row-based view filtering, you can filter a component based on the contents of one of the component's attributes. By having different views on the same item type with different filtering conditions, you can separate the data for an item type into logical blocks, allowing users to view only certain data, depending on which view is used to access the data. Therefore, in query, row-based view filtering helps to automatically limit the amount of data retrieved for a given view.

Caution:

Improperly using row-based view filtering can result in a significant increase in the length of the generated SQL and a decrease in query performance. In the DB2 Content Manager system, your query gets converted to a SQL query string that is executed on the underlying database tables. Since database systems have a limit on the length of the SQL query string, improper usage of filtering can cause this string

to become so long that it can exceed the limit and prevent successful execution of your query. You should review the performance discussion before you decide to use this feature.

Sample usage scenario

This section provides a simple scenario that describes, from a high-level perspective, how row-based view filtering can be used in the DB2 Content Manager system. Following are the steps involved in the scenario:

1. Define an item type called `Journal`.
2. Add some items to this item type.
3. Define an item type view called `MyJournal` with the following filter:
`@Organization = "IBM"`
4. Execute a query against the item type view `MyJournal`.
5. Display the results of the query to the user. Only journals for the IBM organization are returned.

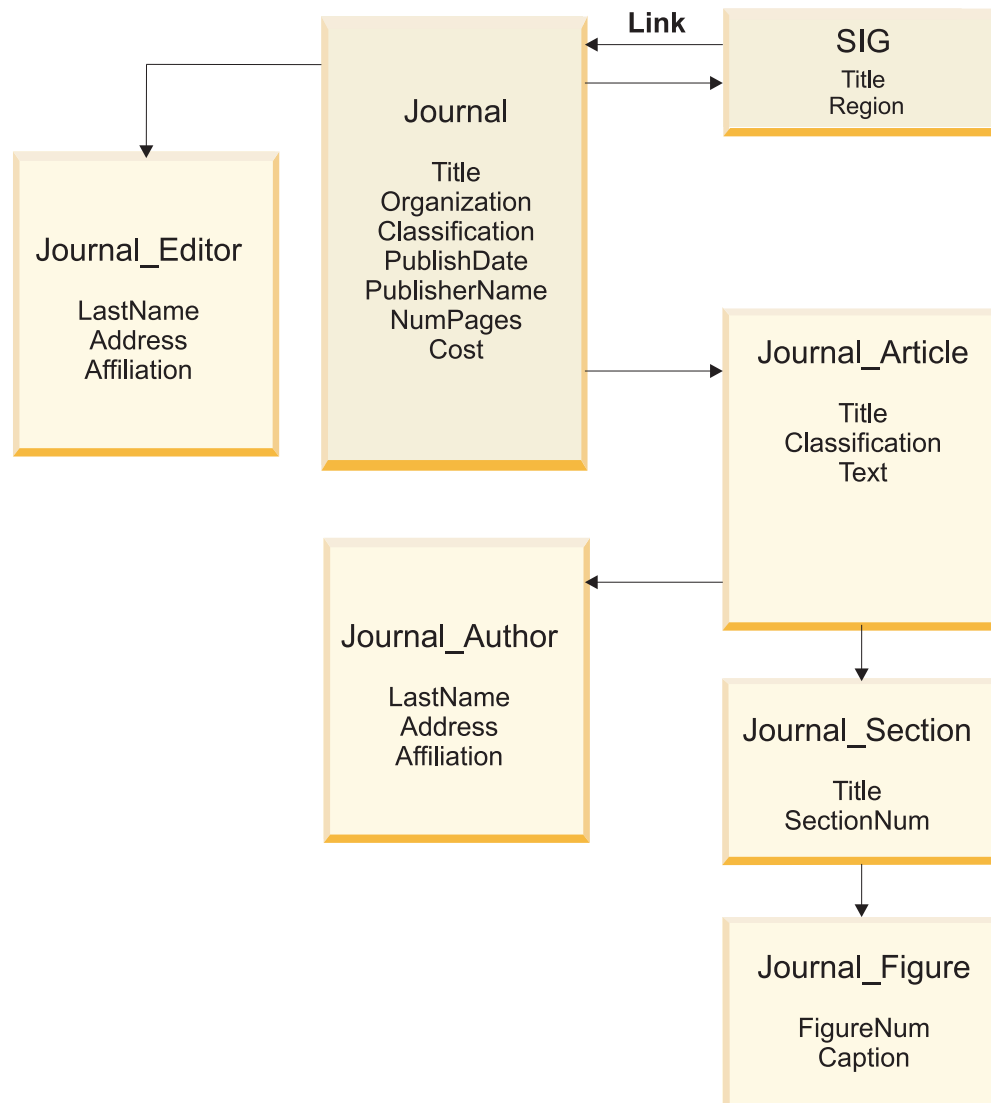


Figure 12. Sample data model for query row-based view filtering

Table 19. Sample definition of views with filters

User Component Type View	Base Component Type	Filter Condition
MyJournal	Journal	@Organization = "IBM"
MyJournal_Article	Journal_Article	@Classification = "Public"
MyJournal_Section	Journal_Section	None
MyJournal_Figure	Journal_Figure	@FigureNum = 5
MyJournal_Author	Journal_Author	@Affiliation = "Almaden"
MyJournal_Editor	Journal_Editor	@Affiliation = "Almaden"
MySIG	SIG	@Region = "USA"

Description of behavior

Row-based filtering in DB2 Content Manager query is applied on component type views explicitly mentioned in the query string or implicitly indicated by a wildcard or a query on one of the view attributes. The following list describes how row-based view filtering behaves depending on the type of query that you want to perform.

Root filters

You can use a filter on the root component type view to filter the contents of the whole item if the root component type view is explicitly mentioned in the query.

Example:

There are 1,000,000 components of the component type Journal in the system. 1,000 of these components have "IBM" in the Organization attribute. You execute the following query to get all journals associated with the view MyJournal: "/MyJournal"

Result: Only 1,000 components are returned from the query since the user query is equivalent to "/MyJournal [@Organization = "IBM"]". The results do not include items for which the row-based filter on Organization does not match the data. For example, items that have Microsoft and Sun in the Organization attribute are not returned. The benefit is that you do not need to specify the filtering conditions explicitly since they are applied automatically based on which view you query.

Important: A component cannot be filtered based on the filters of its children. Filters are not applied down the tree hierarchy. Note that neither MyJournal_Editor's nor MyJournal_Figure's filters are applied for the above "/MyJournal" query.

Child filters

When you use a child component type view in a query, the specific filter for that particular child view is applied.

Example:

You execute the following query to get all journal articles available to an individual: "//MyJournal_Article"

Result: The filter condition @Classification = "Public" is applied on MyJournal_Article. Therefore, only the articles for public consumption are retrieved, but whitepapers and other articles that might not be approved for public viewing are ignored.

Note that a component cannot be filtered based on the filters of its parent. For example, for the query "//MyJournal_Article", only the filter on MyJournal_Article is applied. A filter on the parent view MyJournal is not applied.

If you want a parent filter to be applied, you can take one of the two following approaches:

- Rewrite the query to go through the root, as in this example:
"/MyJournal/MyJournal_Article"
- Add the same filter attribute and filter condition to the child that you added to the root when designing the item type and populate the value for the child filter attribute with the same data as the root filter attribute. For example, you can add the attribute Organization to MyJournal_Article with the following filtering condition: @Organization = "IBM".

Note that a component cannot be filtered based on the filters of its siblings or distant relatives. For example, for the query "/MyJournal_Editor", only the filter on MyJournal_Editor is applied. Filters on MyJournal_Article or on any of its children (MyJournal_Author and MyJournal_Figure) are not applied.

Filters on intermediates

Using the DB2 Content Manager query language, you can traverse through links and references before getting to the final component. Filters are applied on any components involved in such intermediate traversals.

Example:

You execute the following query to get all special interest groups that have links to publications by a specific publisher: "/MyJournal [@PublisherName = "Acme Publishing"]/INBOUNDLINK/@SOURCEITEMREF => MySIG" **Result:** Even though no journals are returned as results of this query, a filtering condition (@Organization = "IBM") is applied on the Journal component type, therefore filtering out any SIGs that have links to journals of other organizations.

Filters on wildcards

Since a wildcard refers to all active component type views for a given user, filtering conditions are applied on any such views if filters are defined on them.

Example:

You execute the following query to find all items that are linked from special interest groups named "XML": "/MySIG [@Title = "XML"]/OUTBOUNDLINK/@TARGETITEMREF => *" **Result:** The wildcard (*) is expanded to represent all active component type views, and filtering conditions are added on MyJournal and MySIG views to ensure that only journals corresponding to IBM and only SIGs in the USA are considered as final results.

Performance considerations

This section contains recommendations for ensuring the best query performance.

Make your filters highly restrictive

A restrictive filter matches only a small portion of the total number of rows for a component type. Using restrictive filters generally leads to better execution plans for your database queries.

Example:

In the earlier example, the filter on MyJournal's Organization attribute is a restrictive filter since only 1,000 components, out of a total of 1,000,000 components, have Organization = "IBM".

Minimize the use of wildcards in your queries

Minimizing the use of wildcards in your queries is always a good idea.

Specifically, since application of each filter generally involves adding extra conditions and joins to your queries, the combination of this extra complexity can be significant when you use wildcards. Because a wildcard refers to all of a user's active views in the system, your query can get large as the number of filtered views increases. Whenever possible, use a specific component type view instead of a wildcard.

Example:

If you know that in your data model SIG items link only to Journals or Books, use the specific names of those component type views in your query instead of the wildcard to indicate the target of link traversal. For example, to find all items that are linked from special interest groups named "XML", rewrite your query as follows: Sub-optimal: `"/MySIG [@Title = "XML"]/OUTBOUNDLINK/@TARGETITEMREF => *`
Optimal: `"/MySIG [@Title = "XML"]/OUTBOUNDLINK/@TARGETITEMREF => MyJournal UNION /MySIG [@Title = "XML"]/OUTBOUNDLINK/@TARGETITEMREF => MyBook"`

Avoid defining too many filters in your system

The more filtered views you have in the system, the more complex your final database queries become when you use wildcards. If you decide to use filters, define these filters only on the views that really require such filtering or avoid using wildcards.

Example:

The simple query `"/* [@ITEMID = "myItemID"]"` to retrieve an item with a specific ITEMID can result in a complex final SQL query if you have, for example, 30 root item type views in the system with 25 of them having filters defined on them. For each filter, extra conditions and joins must be added to your database query. If you know, for example, that your item belongs to either a Journal or a SIG item type, rewrite your query in the following way: `"(Journal | SIG) [@ITEMID = "myItemID"]"`

Since there is a limit on the length of the SQL query string that the database can process, some of your queries might throw an exception if you have a lot of filters defined and you use wildcards.

Database Index on each filtered attribute

This section describes the relationship between database indexes and performance of row-based view filtering queries.

Indexes automatically defined by the DB2 Content Manager system

When a row-based filter is defined on a component type view, the library server automatically tries to create a database index on the column corresponding to the filter attribute. This index should help to improve the performance for complex queries that are written against a data model that uses row-based view filtering. The database system can use the index to come up with a better access plan to execute a complex query.

Example:

If the Organization attribute is specified as a filter attribute during creation of the user view MyJournal on the item type Journal, a database index is created on the column corresponding to the Organization attribute in the table of the base component type for Journal.

An index is created on the filter attribute regardless of whether the component type view in which this attribute exists is a root view or one of the child views. Also, the library server relies on the database system to determine whether an index can be created. In some cases the index might not be created if an index already exists, for example. To verify that an

index has been created on the Organization filter attribute for Journal, for example, a system administrator can go to DB2 Content Manager **System Administration Client -> Data Modeling -> Item Types -> Journal -> Database Indexes** and look for a listing of an index on the Organization attribute.

Indexes that a system administrator can define to improve performance

To improve performance of some wildcard queries on a DB2 Content Manager system that has filtered views, a system administrator can define indexes on the following columns in system tables:

ICMSTITEMS001001.COMPONENTTYPEID
 ICMSTITEMVER001001.COMPONENTTYPEID
 ICMSTRI001001.SOURCECOMPTYPEID

For DB2 Content Manager user component tables (ICMUT* tables) that have reference attributes in them, you can create indexes on the RTARGETCOMPTYPEID column (for the SYSREFERENCEATTRS reference attribute) or any ATTRXXXXX00110 columns (for user-defined reference attributes), such that XXXXX is the attribute group ID of the reference attribute, for example, 01005.

Security implications

Although you can use row-based view filtering to mimic a security mechanism, there are some limitations you should consider before using this feature. It is important to understand exactly how filtering is applied in query and in other parts of the system. For query, there are cases for which the filters are not applied. Besides query, the user might have access to database views that allow them to see filtered data. An application might also be able to retrieve child components directly using other parts of the API if the PIDs of the components are known ahead of time.

The query language grammar

The query language formal grammar, described in the Extended Backus-Naur Form (EBNF) notation, 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");
- IN = ("i" | "I"), ("n" | "N");
- 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 | IN);
- (* literals *)
- DIGIT = ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") ;
- NONZERO_DIGIT = ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") ;
- Exponent = (e | E), ["+" | "-"], DIGIT, { DIGIT }
- INTEGER_LITERAL = "0" | NONZERO_DIGIT, {DIGIT} ;
- FLOAT_LITERAL = DIGIT, { DIGIT }, ".", { DIGIT }, [Exponent]
| ["."], DIGIT, { DIGIT }, [Exponent] ;
- (* UNICODE_CHARACTER is the set of all unicode characters and escape sequences. It's definition is not included in this document *) (* String literals are delimited by double quotes and can contain any character except double quote. To include a double quote as the part of the string literal, specify two consecutive double quotes i.e. a double quote is escaped by another double quote. These will be treated as one double quote character *)
- STRING_LITERAL = "'", { (UNICODE_CHARACTER - "'") | ("'", "'") }, "'";
- (* Escape sequence is a single character delimited by double quotes. To specify a double quote itself as the escape character, specify two consecutive double quotes, as in a double quote is escaped by another double quote. These will be treated as one double quote character. For the complete explanation of the legal values for ESCAPE_CHARACTER, see the *IBM DB2 Universal Database: SQL Reference Volume 1* 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, [NOT], IN, "(", OptionalExpressionList, ")";
- 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;

Chapter 6. Routing a document through a process

DB2 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 DB2 Content Manager transactions.
- Presentation of only the work that the user can access.
- Single audit trail that includes records for document creation, modification, and routing.

For basic document routing concepts and terminology, see the *System Administration Guide*. Also refer to the samples for additional document routing information.

Understanding the document routing process

Document routing consists of processes, work nodes, work lists, and work packages. The system administrator creates the work nodes, processes, and work lists through the system administration client. A process consists of work nodes. Each work node in the process is a separate step in the process. You can create a process that branches out in several directions. The user determines which branch the work node goes to next. The user can choose from a list of possible selections that the system administrator defines. You can define a server exit when you define a work node. You can define server exits for entering a work node, leaving a work node, and to notify the user when the overload limit is reached. When a process is started, a work package is created. The work package is the routing element and contains the attributes of the work. The attributes of the work package consist of the item PID, priority, owner, and so forth.

Collection points are work nodes with additional function. A work package at a collection point node continues to the next work node in the process once the specified number of items of a specified item type exist in the specified folder. Work lists define the work packages assigned to a user. You can have one or more work lists. Each work list can include one or more work nodes. You can specify the order of the work packages in the work list by priority, or date. You can also define the order of work nodes in the work list.

When you retrieve work lists, you can filter the results to include or exclude suspended work. Work packages can also be in notify state. *Notify* state is when work packages have been at the node for longer than the time specified by the administrator. Remember that a work node can be in more than one worklist. The number of packages returned in a work list is defined by the system administrator.

The basic operations you can perform using document routing include:

- Start a process
- End a process
- Continue a process
- Suspend a process

- Resume a process
- Get work from a work list
- Get the next item from a work list
- Define, update, and delete a process
- Define, update, and delete work node
- Define, update, and delete a work list

Understanding document routing enhancements in Version 8.3

DB2 Content Manager features the following document routing enhancements in Version 8.3:

Graphical builder

A new graphical and content-centric workflow builder is included in DB2 Content Manager Version 8.3. You can use it to model your document flows of business processes. To define document routing processes in DB2 Content Manager Version 8.3, you must use the provided graphical workflow builder. Before a document routing process definition can be used to route documents, it needs to be verified by the graphical workflow builder. See the *System Administration Guide* for usage details.

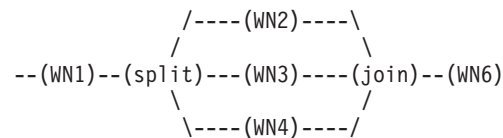
XML of process (diagram) import and export

The graphical builder can import and export a process diagram in its XML format. This feature is useful when you want to transport a diagram from one system to another system. The system administration client can also import and export Content Management objects in XML format. This feature is useful for transporting a process definition, including its related definition objects (such as work nodes) from one system to another system. For more information on XML document routing, see “Accessing DB2 Content Manager document routing using XML-based requests” on page 493.

Parallel routing

Parallel routing is a scheme that can replicate a work package into multiple linked copies and forward them into parallel routes. Parallel routing starts with a split node and ends with a join node. Split-joins work like parentheses in a mathematical expression. It is a one-to-one mapping. You can have nested parallel routes.

The following example illustrates a parallel routing:



1. The work package leaves node (WN1). The split node routes the package to multiple work nodes (WN2, WN3, and WN4).
2. If the work package is updated in WN2, WN3, or WN4, then all three copies mirror the same update.
3. Later in the process, the work package joins together at a single join node before continuing to the next work node (WN6).

No route should ever breach the enclosure within a split-join pair. For example, connecting a route between WN4 and WN6 would be illegal because it would breach the envelope between the split-join pair enclosure. However, you could connect a direct route between WN2 and WN3 because this does not violate the parallel routing enclosure rule.

Decision point

A decision point is a "staffless" worknode in the process that decides which of several possible routes that the work package should proceed through. You can base these decisions on system-defined values, document attributes, and workflow container variables. Whenever a work package reaches a decision point, DB2 Content Manager tests the work package against *decision expressions* associated with the routes. These routes are tested in the order of their assigned precedence, i.e., 1, 2, 3. The work package continues down the first expression that returns a TRUE value. An *otherwise route* (precedence 0) is required for a decision point. A work package cannot be sent down more than one route.

When modeling a decision point node, make sure to correctly define all of the nodes prior to the decision point. Otherwise, a decision could be made on the work node variables defined in workbaskets, collection points, and sub-process nodes (where workbaskets and collection points are part of the process) that cannot be reached prior to the decision point.

When the user displays the decision branch, it lists all of the work node variables that the decision point can be defined for. This list accumulates the variables as the work package traverses from the start node to each connected node prior to decision point node.

Caution: An error can occur if the decision point node cannot read the work node variables that it needs (for example, from a workbasket, collection point, or sub-process). A couple of scenarios can cause this to happen:

- Updating or deleting a workbasket, collection point, or sub-process without updating or deleting its corresponding decision in the decision point node.
- Basing a decision on work node variables that have not been assigned yet. For example, if the work package skipped over certain nodes due to branching or parallel routing.

Business application node (LOB node)

Business application nodes are process steps where the library server invokes a user exit DLL to run lines of business applications. For details about user exits, see "Programming document routing user exits" on page 248.

Workflow action

Using workflow action objects, you can customize workflow applications to integrate DB2 Content Manager with external systems. For example, the workflow action object can refer to an external DLL or Java class to retrieve and e-mail a document. The application integrator is responsible for invoking, validating, and implementing the external DLL or Java class described by the workflow action. DB2 Content Manager never acts on any of the workflow actions in the objects. DB2 Content Manager merely holds the workflow actions in the objects for a separate application to read.

Workflow action list

An actionList represents a collection of workflow actions that you can associate with worknodes in a process. When a work package reaches such a worknode, a client application can retrieve the actionList and show all of the workflow actions that a user can select.

Workflow container variables

Workflow container variables represent instances of name-value pairs (strings) within work packages. The variables serve as helper objects that

can describe data types and expected runtime behavior (for example, name, prompt, and display to user). These helper objects can formulate decision expressions for a decision point. They can also be used by the client to interpret expected behavior for displaying that workflow container variables during runtime. Note that DB2 Content Manager does not enforce any semantic checking or validation for the name-value pairs against their corresponding container variable definitions.

Subprocesses

Subprocesses are processes within another process. After you define a process, you can re-use that same process with another process definition. Subprocess are modeled by using worknode of specific type DK_ICM_DR_SUB_PROCESS_NODE_TYPE.

For further information about the new document routing functionality, see the following sections:

- “Understanding Version 8.3 compatibility with Version 8.2”
- “Understanding document routing classes”
- “Document routing constants” on page 251

Understanding Version 8.3 compatibility with Version 8.2

Document routing processes created in Version 8.2 will continue to work in Version 8.3. After updating your Version 8.2 library server to Version 8.3, you can use the graphical builder to modify existing processes that were created using Version 8.2 APIs or Version 8.2 system administration client. The graphical builder displays those previously defined Version 8.2 processes graphically and gives you the option to rearrange the diagram layout. After re-arranging the diagram layout, you need to re-verify the diagram before you can save it.

If you make modifications to existing Version 8.2 processes or create new processes that take advantage of Version 8.3 functionality (i.e. parallel routing, business application node, decision point, and subprocess), you must update your clients to Version 8.3. Version 8.2 clients will not work with Version 8.3 library server that contains processes taking advantage of Version 8.3 document routing enhancements.

Understanding document routing classes

There are 12 classes that you can use to implement document routing functionality into your application. You can find the details about these classes and methods in the *online API reference*. The document routing APIs include:

DKDocRoutingServiceICM

This class provides the methods for routing and accessing workpackages and container data through a process. For instance, start, terminate, continue, suspend, resume, listWorkPackages, and setWorkPackageContainerData.

DKDocRoutingServiceMgmtICM

This class provides the methods to manage the document routing definition classes: DKProcessICM, DKWorkNodeICM, and DKWorkListICM, DKWorkflowActionICM, DKWorkflowActionListICM.

Retrieving a copy of the DKDocRoutingServiceMgmtICM object from the DKDocRoutingServiceICM object is more efficient than creating a DKDocRoutingServiceMgmtICM object of your own because the API can internally share the same copy of workflow definition data.

DKProcessICM

This class represents a process definition which contains a collection of interconnecting routes that describe the steps and flows of a process (in addition to other attributes such as timelimit) and description of a process.

It is possible to create Version 8.3 document routing process definitions with API directly. But you should avoid doing so if at all possible. Because creating document routing process with the Version 8.3 APIs risks unexpected behavior (such as creating illegal parallel routing construct) and damage to the system, the graphical workflow builder averts this by validating a process before it is saved into the library server.

The capability to save the process relies on the definition of three premises:

Process state

Indicates whether the process has been verified or is still in draft state. You can retrieve this status with the `getState` method in `DKProcessICM`. This returns either `DK_ICM_DR_PROCESS_DRAFT_STATE` (0) or `DK_ICM_DR_PROCESS_VERIFIED_STATE` (1). Although you can toggle this status with the `setState` method, you should allow the graphical workflow builder to handle this to avoid any damage to the system.

Diagram definition

Defines the graphical appearance of the process in an array of bytes. The maximum length is 1 Mb. The two `DKProcessICM` methods: `setDiagramDefinition(byte[] diagram_definition)` and `getDiagramDefinition(byte[] diagram_definition)` set and get the diagram definition.

Route definition

Defines the route. The extended `DKRouteListEntryICM` connects the following types of work nodes:

- Workbasket: `DK_ICM_DR_WB_NODE_TYPE` (0),
- Collection point node: `DK_ICM_DR_CP_NODE_TYPE` (1)
- Business application node: `DK_ICM_DR_BA_NODE_TYPE` (2)

`DKRouteListEntryICM` also connects the following virtual nodes:

- Split node: `DK_ICM_DR_SPLIT_NODE_TYPE` (3)
- Join node: `DK_ICM_DR_JOIN_NODE_TYPE` (4)
- Decision point node: `DK_ICM_DR_DP_NODE_TYPE` (5)
- Subprocess node: `DK_ICM_DR_SUB_PROCESS_NODE_TYPE` (6)
- Start node: `DK_ICM_DR_START_NODE_TYPE` (7)
- End node: `DK_ICM_DR_END_NODE_TYPE` (8)

Virtual nodes are used by the system to facilitate process navigation. They are called virtual nodes because the document routing APIs do not return work packages for them. You can, however, query for the location of work packages (in their raw DDO format).

The `DKRouteListEntryICM` methods: `setDecisionRuleExternal`, `getDecisionRuleExternal`, `setDecisionRuleInternal`, `setPrecedence`, and `getPrecedence` set date for routes originating from decision points. `DecisionRuleExternal` points are used by the graphic workflow builder to display decision rules to users;

DecisionRuleInternal points are used by the library server to evaluate decision rule outcome during runtime.

The listProcessNames() and listProcesses() methods in DKDocRoutingServiceMgmtICM just list verified processes. Processes in draft state will not be returned.

The add(DKProcessICM process) and update(DKProcessICM process) methods can only save processes that passed the graphic workflow builder verification. Therefore, the add and update methods only work in the following scenarios:

- The process is in the draft state, and the route list is undefined; but the diagram definition is defined.
- The process is in the verified state; the diagram is undefined; and the route list is defined.
- The process is in the verified state, and both the diagram and the route list are defined.

In these scenarios, the graphic workflow builder will regulate the proper combinations of process attributes before it is saved in order to ensure system data correctness. The process cannot be saved for any other combination. See Table 20 for all possible combinations.

Table 20. Combinations that the add(DKProcessICM process) and update(DKProcessICM process) method can save

Process state	Diagram definition	Route definition	Can be saved
Draft (0)	No	No	No
Draft (0)	No	Yes	No
Draft (0)	Yes	No	Yes
Draft (0)	Yes	Yes	No
Verified (1)	No	No	No
Verified (1)	No	Yes	Yes
Verified (1)	Yes	No	No
Verified (1)	Yes	Yes	Yes

DKWorkNodeICM

This class represents a work node definition which details the expected or applicable tasks (such as Workflow ActionList and library server exit) to perform at that particular step in a process.

Note that a subprocess is recorded as a work node of type DK_ICM_DR_SUB_PROCESS_NODE_TYPE. The subprocess and the worknode share the same name.

DKWorkNodeContainerDefICM

This class serves as a helper that describes the intended data type and expected runtime behavior (such as name,"prompt, and display to user) for workflow container data routed along a workpackage.

Note that the actual container data are instances of name-value pair (type string). The instances of container data are child components of workpackage instances, not instances of this DKWorkNodeContainerDefICM class. DB2 Content Manager does not

enforce any semantic checking and validation for those name-value pairs against their corresponding container variable definitions.

DKWorkflowActionICM

This class records details (such as file name and function name) of an external DLL or Java class to be run at the API client side. DB2 Content Manager does not automatically invoke the exit that you specify in the DKWorkflowActionICM object. A custom application must invoke, validate, and implement the workflow action.

DB2 Content Manager pre-defines several workflow actions. They primarily help IBM clients (such as eClient and pClient) display available menu selections at a worknode. Note that DB2 Content Manager does not automate or enforce any of these system-defined workflow actions.

DKWorkflowActionListICM

This class records a collection of workflow actions and its name is returned along with the retrieval of a worknode. This enables client application to display applicable workflow actions at a worknode.

DKWorkListICM

This class represents a filtered view (based on criteria such as worknode and priority) for a collection of workpackages that route documents to which users have access.

DKRouteListEntryICM

This class defines the route that a process can take (as in from or to). A process object (DKProcessICM) contains a collection of route entry objects (DKRouteListEntryICM).

Certain fields (decision rule and precedence) facilitate the construction of decision rules for routes that leave the decision nodes.

Recommendation: You can write a program that uses the APIs to retrieve and examine the value set by the graphical builder, but do not set those decision rules by yourself for risk of creating an invalid decision rule.

DKCollectionResumeListEntryICM

This class represents an entry in the resume list for the collection point work nodes.

DKResumeListEntryICM

This class represents an entry of resume list to set whenever a suspended work package waits for the arrival of certain documents (not necessarily at a collection point) in order to resume.

DKWorkPackageICM

This class represents a work package for a routing task. It contains a persistent identifier for the document to route, and information on the routing state (such as process name, worknode name, and completion time).

When a process starts, one or more work packages are created as a result of the API call.

Creating document routing service objects

The examples below demonstrate how to create a document routing object.

Java

```
// The DKDocRoutingServiceICM class provides the core routing services
// like starting, terminating, continuing, suspending, and resuming a process.
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);

// 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
//
// Retrieving the DKDocRoutingServiceMgmtICM from DKDocRoutingServiceICM has
// the benefit of reducing cache footprint by sharing definition objects
// between these two classes.
DKDocRoutingServiceMgmtICM routingMgmt=
    routingService.getDocRoutingServiceMgmt();
```

C++

```
// The DKDocRoutingServiceICM class provides the core routing services
// like starting, terminating, continuing, suspending, and resuming
// a process.
DKDocRoutingServiceICM* routingService = new
    DKDocRoutingServiceICM(dsICM);

// 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
//
// Retrieving the DKDocRoutingServiceMgmtICM from DKDocRoutingServiceICM
// has the benefit of reducing cache footprint by sharing definition
// objects between these two classes.
//
// Deleting routingService object will also clean up memory allocated
// for routingMgmt
DKDocRoutingServiceMgmtICM* routingMgmt =
    routingService->getDocRoutingServiceMgmt();
```

For a complete sample, refer to the SDocRoutingDefinitionCreationICM sample.

Defining a new regular work node

A work node is a step in a document routing process definition. It may be completed to continue to the next step by a customer application at any time. The application is responsible for validating its own completion criteria.

Java

```
// Create new Work Node Object.
DKWorkNodeICM workNode1 = new DKWorkNodeICM();
//Choose a Name that is 15 characters or less length
workNode1.setName("S_fillClaim");
//Choose a Description for more information than the name.
workNode1.setDescription("Claimant Fills Out Claim");
// Sets the value of the maximum time that an item can spend at this
// work node (in minutes).
workNode1.setTimeLimit(100);
// Specify the threshold that activates overload user
// exit when the number of work packages at this node
// is greater than the threshold.
// Note that the number of workpackages at this worknode
// is not limited to the threshold.
workNode1.setOverloadLimit(200);
workNode1.setOverloadUserDll("C:\\USEREXIT\\exitOverload.dll");
workNode1.setOverloadUserFunction("callOverload");

// Set the type to be a regular work node
workNode1.setType(0);

//Add the new work node definition to the document routing
//managment object
routingMgmt.add(workNode1);
```

C++

```
// Create new Work Node Object.
DKWorkNodeICM* workNode1 = new DKWorkNodeICM();
// Choose a Name that is 15 characters or less length
workNode1->setName("ValidateCreditCard");
// Choose a Description for more information than the name.
workNode1->setDescription("Buyer's credit card is validated with
                        the credit card agency");

// Sets the value of the maximum time that an item can spend at
//this work node (in minutes).
workNode1->setTimeLimit(100);

// Specify the threshold that activates overload user
// exit when the number of work packages at this node
// is greater than the threshold.
// Note that the number of workpackages at this worknode
// is not limited to the threshold.
workNode1->setOverloadLimit(200);
workNode1->setOverloadUserDll("C:\\USEREXIT\\exitOverload.dll");
workNode1->setOverloadUserFunction("callOverload");

// Set the type to be a regular work node
workNode1->setType(0);

//Add the new work node definition to the document routing
//management object
routingMgmt->add(workNode1);

// Free memory. This object will no longer be needed.
delete(workNode1);
```

For a complete example creating work notes, refer to the SDocRoutingDefinitionCreationICM sample.

Listing work nodes

The `listWorkNodeNames` method lists work node names in the library server. If no specific worknode type parameter is requested then all work nodes of type work basket, collection point, and business application node are listed by default.

The `listWorkNodes` method returns a collection of `DKWorkNodeICM` objects representing work nodes in the library server.

Java

```
// Obtain the document routing management object.
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt = new
    DKDocRoutingServiceMgmtICM(dsICM);

// Obtain all Work Nodes in the System.
dkCollection workNodes = routingMgmt.listWorkNodes();
System.out.println("Work Nodes in System: (" + workNodes.cardinality() + ")");
dkIterator iter = workNodes.createIterator();
while(iter.more())
{
    DKWorkNodeICM workNode = (DKWorkNodeICM) iter.next();
    if(workNode.getType() == 0)
        System.out.println(" Normal Node - " + workNode.getName() + ": "
            + workNode.getDescription());
    else
        System.out.println(" Collection Pt - " + workNode.getName() + ": "
            + workNode.getDescription());
}
```

C++

```
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM* routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the collection containing all the work nodes in the system.
dkCollection* workNodes = routingMgmt->listWorkNodes();
if (workNodes && (workNodes->cardinality()>0) )
{
    cout << "Work Nodes in System: (" << workNodes->cardinality() << ")"
    << endl;
    dkIterator* iter = workNodes->createIterator();
    while(iter->more())
    {
        DKWorkNodeICM* workNode = (DKWorkNodeICM*)iter->next()->value();
        if(workNode->getType()==0)
        {
            cout << " Normal Node - "
            << workNode->getName() << ": " <<
            workNode->getDescription() << endl;
        }
        else
        {
            cout << " Collection Pt - "
            << workNode->getName() << ": " <<
            workNode->getDescription() << endl;
        }
        delete(workNode);
    }
    delete(iter);
    delete(workNodes);
}
delete(routingMgmt);
```

For more information on listing work nodes, refer to the `SDocRoutingListingICM` sample.

Defining a new collection point

A collection point is a work node that enforces document availability as the completion criteria (specifically applicable to routing folders). In a collection point, you can specify requirements to meet before the process can resume or advance past this point by the system. Alternatively, you can call `continueProcess` to force a work package to move out of the collection point without satisfying the document availability requirements.

Java

```
// Create a new Work Node Object. This will be
//the collection point
DKWorkNodeICM collectionPoint = new DKWorkNodeICM();

// Choose a Name with 15 characters or less.
collectionPoint.setName("S_gatherAll");

// Choose a Description for more information than the name.
collectionPoint.setDescription("Gather Claim,Police Report,Policy,& Photos");

// Sets the value of the maximum time that an item can spend at this
// work node (in minutes).
collectionPoint.setTimeLimit(100);

// Specify the threshold that activates overload user
// exit when the number of work packages at this node
// is greater than the threshold.
// Note that the number of workpackages at this worknode
// is not limited to the threshold.
collectionPoint.setOverloadLimit(200);
collectionPoint.setOverloadUserDll("C:\\USEREXIT\\exitOverload.dll");
collectionPoint.setOverloadUserFunction("callOverload");

// 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
// management object
routingMgmt.add(collectionPoint);
```

C++

```
// Create a new Work Node Object. This will be the collection point
DKWorkNodeICM* collectionPoint = new DKWorkNodeICM();
// Choose a Name with 15 characters or less.
collectionPoint->setName("GatherOrderDetails");
// Choose a Description for more information than the name.
collectionPoint->setDescription("Gather all the information related to the
    order, shipping mechanism and shipping address");
// Sets the value of the maximum time that an item can spend at this
//work node (in minutes).
collectionPoint->setTimeLimit(100);

// Specify the threshold that activates overload user
// exit when the number of work packages at this node
// is greater than the threshold.
// Note that the number of workpackages at this worknode
// is not limited to the threshold.
collectionPoint->setOverloadLimit(200);
collectionPoint->setOverloadUserDll("C:\\USEREXIT\\exitOverload.dll");
collectionPoint->setOverloadUserFunction("callOverload");

// 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);

// continued...
```

C++ (continued)

```
// 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
// management object
routingMgmt->add(collectionPoint);

//Free the memory associated with this collection point
// Note that the resumeRequirement collection becomes a part of the
// collectionPoint object once the affinity between those two objects
// are established. Deleting the collectionPoint object cleans up
// memory allocated for resumeRequirement object.
delete(collectionPoint);
```

For more information on defining collection points, refer to the `SDocRoutingDefinitionCreationICM` sample.

Defining a work list

A worklist consists of one or more work nodes from which a user obtains a list of work packages or the "next" work package. A work node can be in more than one worklist. Using a worklist allows an administrator or a user application to dynamically change work assignments without contacting end users.

Java

```
// Create a new work list.
DKWorkListICM workList = new DKWorkListICM();
// Choose a name of 15 characters or less.
workList.setName("S_fillClaimWL");
workList.setDescription("Work List Covering Fill/Submit Claim Work Node.");
//Specify that work packages returned by the work list will be sorted by time
workList.setSelectionOrder(DKConstantICM.DK_ICM_DR_SELECTION_ORDER_TIME);
//Specify that the work packages returned will be the one that are not
// in the suspend state
workList.setSelectionFilterOnSuspend
(DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList.setSelectionFilterOnNotify
(DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

// Specify that at most 100 work packages should be listed in
// this work list
workList.setMaxResult(100);
String[] wnNames = {"S_fillClaim"};
workList.setWorkNodeNames(wnNames);

//Add the new work list definition to the document routing
//management object
routingMgmt.add(workList);
```

C++

```
// Create a new worklist.
DKWorkListICM* workList = new DKWorkListICM();
//Choose a name of 15 characters or less.
workList->setName("ValidateWorkList");
workList->setDescription("worklist Covering the credit card
    validation work node.");

//Specify that work packages returned by the worklist will be
//sorted by time
workList->setSelectionOrder(DK_ICM_DR_SELECTION_ORDER_TIME);

//Specify that the work packages returned will be the one that are not
//in the suspend state
workList->setSelectionFilterOnSuspend(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList->setSelectionFilterOnNotify(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that at most 100 work packages should be listed in this worklist
workList->setMaxResult(100);

DKString* wnNames = new DKString[1];
wnNames[0] = "ValidateCreditCard";
//Add the work node to the worklist
workList->setWorkNodeNames(wnNames,1);

//Add the new worklist definition to the document routing management
//object
routingMgmt->add(workList);

//Free the memory associated with this worklist
delete(workList);
```

For more information on defining work lists, refer to the `SDocRoutingDefinitionCreationICM` sample.

Listing worklists

The `listWorkListNames` method lists names of worklists to which the logged in user has access. The `listWorkLists` method returns a collection of `DKWorkListICM` objects representing worklists in the library server.

Java

```
// Obtain the document routing management object.
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain all Work Lists in the System.
dkCollection workLists = routingMgmt.listWorkLists();
System.out.println("Work Lists in System: (" +workLists.cardinality()+)");
dkIterator iter = workLists.createIterator();
while(iter.more())
{
    DKWorkListICM workList = (DKWorkListICM) iter.next();
    System.out.println("    - " +workList.getName()+":
        "+workList.getDescription());
}
```


C++

```
dkCollection* workLists = routingMgmt->listWorkLists(); // Obtain all
// Work Lists in the System.

if (workLists && (workLists->cardinality()>0) )
{
    cout<<"Work Lists in System: ("<< workLists->cardinality()<<")"<<endl;
    dkIterator* iter = workLists->createIterator();
    while(iter->more()){
        DKWorkListICM* workList = (DKWorkListICM*) iter->next()->value();
        cout << " - " << workList->getName() << ": "
            << workList->getDescription() << endl;
        delete(workList); // Free Memory
    }
    delete(iter); // Free Memory
    delete(workLists);
}
```

For the complete example, see the SDocRoutingListingICM sample.

Defining a new process and associated route

A document routing process consists of defined routes that a work package follows. Multiple routing processes can reuse the same nodes and you can also use multiple routes between nodes.

Caution: Although you can create Version 8.3 document routing process definitions with the APIs, you should avoid doing this for risk of damaging the system with unexpected behavior (such as creating an illegal parallel routing construct). The graphical workflow builder averts this by validating a process before saving it into the library server.

Java

```
// Create a new Process Definition
DKProcessICM process = new DKProcessICM();
process.setName("S_claimProcess");
process.setDescription("Process for an Insurance Claim");

// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries. A simple route
//between two work nodes is specified by associating a 'From' work
//node and a 'To' work node.
// A Route List Entry simply connects two nodes with an implied direction.
// Multiple routes may exist between nodes. A specific route may be selected
// by a user-defined "selection" keyword. Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", etc.
// Create a new connection between two nodes.
DKRouteListEntryICM nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute.setFrom(DKConstantICM.DK_ICM_DR_START_NODE);
nodeRoute.setTo("S_fillClaim");
// Choose any user-defined name for an action that will make the
//transition from the 1st node to the second
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_fillClaim");
nodeRoute.setTo("S_gatherAll");
// Choose any user-defined name for an action that will make the
// transition take place.
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_gatherAll");
nodeRoute.setTo(DKConstantICM.DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make
//the transition take place.
nodeRoute.setSelection("Complete");
// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
// Set the route in the process.
process.setRoute(routes);
// Add the process to the routing Management.
routingMgmt.add(process);
```

C++

```
//A document routing process is the defined routes that a work
//package being routed will follow. Multiple routing processes may
//re-use the same nodes and multiple routes between nodes may be used.

// Create a new Process Definition
DKProcessICM* process = new DKProcessICM();
process->setName("Buy_Book");
process->setDescription("Purchase a book online");
// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection* routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries. A simple route
// between two work nodes is specified by associating a 'From' work node
// and a 'To' work node.
// A Route List Entry simply connects two nodes with an implied direction.
// Multiple routes may exist between nodes. A specific route may be selected
// by a user-defined "selection" keyword. Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", etc.

// Create a new connection between two nodes.
DKRouteListEntryICM* nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute->setFrom(DK_ICM_DR_START_NODE);
nodeRoute->setTo("ValidateCreditCard");
// Choose any user-defined name for an action that will make the transition
// from the 1st node to the 2nd
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("ValidateCreditCard");
nodeRoute->setTo("GatherShippingDetails");

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("GatherOrderDetails");
nodeRoute->setTo(DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Complete");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

// Set the route in the process.
process->setRoute(routes);

// Add the process to the routing Management.
routingMgmt->add(process);

delete(process);
```

For the complete example, see the SDocRoutingDefinitionCreationICM sample.

Starting a document routing process

The startProcess method starts a process with the name, item PID, priority, and owner name that you specify. It returns a PID of the work package created by this method. The work package begins at the first work node of the given process.

The example below shows you how to start a document routing process.

Java

```
//First create a document or folder that will be routed.
//An item type of name "S_simple" must be pre-defined before a
// DDO of that name can be created.

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

// Save the created folder to the persistent datastore.
ddoFolder.add();

//Create the core document routing service object.
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "S_claimProcess" which must be pre-defined.
// The PID string of the work package as a result of the startProcess API
// call is returned.
// Note that the PID string may be an empty string if multiple
// workpackages are created by parallel routing.
String workPackagePidStr = routingService.startProcess
    ("S_claimProcess", ddoFolder.getPidObject().pidString(),1,"icmadmin");
```

C++

```
//First create a document or folder that will be routed.
//An item type of name "book" must be pre-defined before a DDO of
// that name can be created.
DKDDO* ddoFolder = dsICM->createDDO("book", DK_CM_FOLDER);

// Save the created folder to the persistent datastore.
ddoFolder->add();

//Set the priority for this document routing process
int Priority = 1;

//Create the core document routing service object.
DKDocRoutingServiceICM* routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "Buy_Book" (which must be pre-defined.
// The PID string of the work package as a result of the startProcess API
// call is returned.
// Note that the PID string may be an empty string if multiple
// workpackages are created by parallel routing.
DKString workPackagePidStr=routingService->startProcess("Buy_Book",
    ((DKPidICM*)
    ddoFolder->getPidObject())->pidString(), Priority, "icmadmin");
```

For the complete example, see the SDocRoutingProcessingICM sample.

Ending a process

You can explicitly terminate a process before it reaches the end node by specifying its work package PID in the `terminateProcess` method. When you terminate a process, all work packages in the process (including workpackages on parallel routes and from subprocesses) are removed from the system.

The method also checks in the item referenced in the work package if the item was checked out.

Java

```
routingService.terminateProcess(workPackagePidStr);
```

C++

```
routingService->terminateProcess(workPackagePidStr);
```

For more information on terminating a process, refer to the `SDocRoutingProcessingICM` sample.

Continuing a process

The `continueProcess()` method routes the item referenced by the item PID in the specified work package from the current work node to the next work node that is determined by the selection. The specified work package is removed from the library server, and a new work package is created for the specified owner. The item referenced by the item PID stays checked out if it has been checked out.

The PID of the new work package is returned. In the Java API, a null string is returned if the process has ended, or if the process continued onto a split node (parallel routes). In the C++ API, an empty string is returned if the process has ended, or if the process continued onto a split node (parallel routes).

In the code snippet below, the name of the selection that will cause the transition from the current work node to the next work node is "Continue". Note that the work package PID string of the current work package is specified in the method call. The method call either returns the PID string of the new work package, or an empty string.

Java

```
workPackagePidStr = routingService.continueProcess  
(workPackagePidStr, "Continue", "icmadmin");
```

C++

```
char * userName = "icmadmin";  
  
workPackagePidStr = routingService->continueProcess(workPackagePidStr,  
"Continue", userName);
```

For the complete example, see the `SDocRoutingProcessingICM` sample.

Suspending a process

The `suspendProcess` method can suspend an instance of a document routing work package for a specific duration (in minutes), or for a given resume list. The method sets the suspend flag of the specified work package to true.

The duration specifies how long to keep the suspend flag at true. The resume list is a set of requirements that instructs a folder to wait for the arrival of certain item types and quantities (specified in a sequential collection of `DKResumeListEntryICM` objects). When the duration elapses, or when the resume list is satisfied, then the system resets the work package `suspendState` flag from true (1) to false (0). You can also force the work package to prematurely move to the next work node by calling the `continueProcess` API.

Suspending a package does not affect the state of other packages on the same process.

The `suspendProcess` API does not relate to processes and threads in a programming environment. The thread or process in the C++ runtime environment will not be stopped.

The `DOCROUTINGUPDATE` field in the system control table controls the scheduled time to re-evaluate the suspend flag of a work package. The default time interval is 10 minutes.

Java

```
dkCollection requirements = new DKSequentialCollection();
//Process will be suspended for 2 minutes.
routingService.suspendProcess(workPackagePidStr, 2, requirements);
```

C++

```
dkCollection * requirements = new DKSequentialCollection();
//If no requirements are to be provided and the process is only to be
//suspended for a fixed period of time, the user can also pass in a
//NULL collection to this method.
//dkCollection * requirements = NULL;

//Process will be suspended for 2 minutes.
routingService->suspendProcess(workPackagePidStr, 2, requirements);
delete(requirements);
```

For the complete example, see the `SDocRoutingProcessingICM` sample.

Resuming a process

A process in the suspended state can resume in three ways:

- Implicitly after the specified time expired.
- Implicitly after the defined requirements have been met.
- Explicitly through the `resumeProcess` method. This method resumes the work package before the specified duration elapses or before resume list is satisfied.

The resume resets the work package suspend flag to false and returns the process to normal operation. No routing or checkout of the associated work item is performed.

Java

```
routingService.resumeProcess(workPackagePidStr);
```

C++

```
routingService->resumeProcess(workPackagePidStr);
```

For more information on resuming a process, refer to the SDocRoutingProcessingICM sample.

Listing work package persistent identifier strings in a worklist

The listWorkPackagePidStrings method returns the work package PIDs of all work packages in the specified work list. Based on the setting in the system administration for the work list, the owner field in the listWorkPackagePidStrings method can return different lists of work packages as shown in Table 21.

Table 21. listWorkPackagePidStrings API results based on owner

System administration setting for a work list	owner=empty	owner=logged on user ID	owner = a user ID other than the logged on user ID
Filter on owner is not checked	Returns all work packages in the work list.	Returns all work packages in the work list.	Returns work packages of the specified user ID in the work list if the ICM_PRIV_ITEM_GET_ASSIGN_WORK privilege is set.
Filter on owner is checked	Returns work packages of the logged on user ID in the work list.	Returns work packages of the logged on user ID in the work list.	Returns work packages of the specified user ID in the work list if the ICM_PRIV_ITEM_GET_ASSIGN_WORK privilege is set.

The following code sample shows how to list the PID strings for all the work packages in a specified worklist.

Java

```
String[] workPackagePIDs =
    routingService.listWorkPackagePidStrings(workListName,processOwner);

// Print Work Package PIDs
System.out.println("Work Packages in Work List: (+workPackagePIDs.length+));
for(int i=0; i< workPackagePIDs.length; i++)
    System.out.println(" - PID: +workPackagePIDs[i]);
```

C++

```
long arraySize = -1; // Size to be set by the API.
DKString* workPackagePIDs = routingService->
    listWorkPackagePidStrings("workListName",processOwner,arraySize);

// Print Work Package PIDs
cout << "Work Packages in Work List: (" << arraySize << ")" << endl;
for(int i=0; i< arraySize; i++)
    cout << " - PID: " << workPackagePIDs[i] << endl;

delete[] workPackagePIDs; // Free Memory
```

For the complete example, see the SDocRoutingListingICM sample.

Retrieving work package information

When an instance of a document routing process is in progress, a work package is the vehicle through which an item (instance of an item type) moves along through the routing process. A work package contains all the necessary information about the process and about the item that it is transporting. The work package is the object that an application uses and manipulates as required.

The `retrieveWorkPackage` method returns the `DKWorkPackageICM` object referenced by the specified work package PID (`wpPidStringStr`).

Java

```
//Use an established document routing service
//Specifying false in this method call makes sure that the work package
// is not checked out
DKWorkPackageICM workPackage =
    routingService.retrieveWorkPackage(workPackagePidStr,false);
System.out.println("-----");
System.out.println("                Work Package");
System.out.println("-----");
System.out.println(" Process Name: " + workPackage.getProcessName());
System.out.println(" work Node Name: " + workPackage.getWorkNodeName());
System.out.println("      Owner: " + workPackage.getOwner());
System.out.println("   Priority: " + workPackage.getPriority());
System.out.println(" User Last Moved: " + workPackage.getUserLastMoved());
System.out.println(" Time Last Moved: " + workPackage.getTimeLastMoved());
System.out.println(" Suspend State: " + workPackage.getSuspendState());
System.out.println("  Notify State: " + workPackage.getNotifyState());
System.out.println("   Notify Time: " + workPackage.getNotifyTime());
System.out.println("   Resume Time: " + workPackage.getResumeTime());
System.out.println("Work Package Pid: " + workPackage.getPidString());
System.out.println("   Item Pid: " + workPackage.getItemPidString());
```


C++

```
cout << "-----" << endl;
cout << " Work Package" << endl;
cout << "-----" << endl;
cout << " Process Name: " << workPackage->getProcessName() << endl;
cout << " work Node Name: " << workPackage->getWorkNodeName() << endl;
cout << " Owner: " << workPackage->getOwner() << endl;
cout << " Priority: " << workPackage->getPriority() << endl;
cout << " User Last Moved: " << workPackage->getUserLastMoved() << endl;
cout << " Time Last Moved: " << workPackage->getTimeLastMoved() << endl;
cout << " Suspend State: " << workPackage->getSuspendState() << endl;
cout << " Notify State: " << workPackage->getNotifyState() << endl;
cout << " Notify Time: " << workPackage->getNotifyTime() << endl;
cout << " Resume Time: " << workPackage->getResumeTime() << endl;
cout << "Work Package Pid: " << workPackage->getPidString() << endl;
cout << " Item Pid: " << workPackage->getItemPidString() << endl;
```

For the complete example, see the SDocRoutingProcessingICM sample.

Listing document routing processes

The following example shows you how to list document routing processes.

Important: The `listProcessNames()` and `listProcesses()` methods in `DKDocRoutingServiceMgmtICM` now just list verified processes by default, unless the process state information is explicitly requested. Processes in draft state are not returned by default. For details on verified processes, see “Understanding Version 8.3 compatibility with Version 8.2” on page 224.

Java

```
//The listProcessNames method lists all process names in the
//Library Server, and the listProcesses method returns a collection
//of DKProcessICM objects representing a process in the Library Server.
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain the list of all running document routing processes
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain list of all routing processes running.
dkCollection processes = routingMgmt.listProcesses();
System.out.println("Running Processes: (" + processes.cardinality() + ")");

dkIterator iter = processes.createIterator();
while(iter.more())
{
    // Move pointer to next element and obtain that next element.
    DKProcessICM proc = (DKProcessICM) iter.next();
    System.out.println(" - " + proc.getName() + ": " + proc.getDescription());
}
```

C++

```
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM* routingMgmt =
    new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the list of all running document routing processes
dkCollection* processes = routingMgmt->listProcesses();
if (processes && (processes->cardinality()>0))
{
    cout << "Running Processes: (" << processes->cardinality() << ")"
    << endl;
    dkIterator* iter = processes->createIterator();
    while(iter->more())
    {
        DKProcessICM* proc = (DKProcessICM*) iter->next()->value();
        cout << " - " << proc->getName() << ": "
        << proc->getDescription() << endl;
        delete(proc);
    }
    delete(iter);
    delete(processes);
}
delete(routingMgmt);
```

A print function is provided in the SDocRoutingListingICM sample.

Ad hoc routing

Below is an ad hoc routing example procedure. In the example, the system administration client is used to set up the work nodes, processes, and worklists.

1. Create two work nodes, N1 and N2 for example.
2. Create two one-node processes, P1 and P2 such that P1 has one work node, N1 and P2 has one work node, N2.

P1 looks like this:

From:	Action:	To:
START	Continue	N1
N1	Continue	END

P2 looks like this:

From:	Action:	To:
START	Continue	N2
N2	Continue	END

3. Create two worklists, WL1 and WL2 such that WL1 has one work node, N1 and WL2 has one work node, N2.

At run time, complete the following steps to implement ad-hoc routing:

1. Start process P1 with a document PID (Example, ABC). A work package, WP1, is created. The worklist WL1 displays the work package WP1 at work node N1.
2. To move the document ABC from process P1 to process P2, terminate work package WP1 and start process P2 with the same document (ABC). Work package WP2 is created.

The worklist WL2 shows the work package WP2 at work node N2.

To see additional examples, see the `SDocRoutingDefinitionCreationICM` sample.

Document routing example queries

This sections contains example queries. For more information about writing queries, see the section on Searching for data.

Example 1

Returns car documents whose associated work packages are active (not suspended).

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =  
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@SUSPENDFLAG =  
0]
```

Example 2

Returns car documents whose associated work packages are in the AccidentInvestigation process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =  
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =  
/ROUTINGPROCESS[@PROCESSNAME =  
"AccidentInvestigation"]/@ITEMID]
```

Example 3

Returns car documents where the name is Honda, and documents' associated work packages are in the AccidentInvestigation process.

```
/Car[@Name = "Honda" AND @VERSIONID = latest-version(.) AND  
@SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER =>  
WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME =  
"AccidentInvestigation"]/@ITEMID]
```

Example 4

Returns car documents whose associated work packages 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

Returns car documents whose associated work packages are suspended in the UnderReview step of the AccidentInvestigation process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =  
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =  
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]  
/@ITEMID AND ../@WORKNODENAME = "UnderReview" AND  
@SUSPENDFLAG = 1]
```

Granting privileges for document routing

In order for a user to perform document routing operations, the user must have the appropriate privileges. The privileges associated with document routing are listed in the following table. The general privileges for items are applicable to processes, work nodes, and worklists.

Table 22. Document routing privileges

Privilege	Description	Related API
ICM_PRIV_ITEM_UPDATE_WORK	Used to see if the user is authorized to do the following for a work package: set the priority set the owner set the resume list set the duration for suspension	suspendProcess resumeProcess setWorkPackagePriority setWorkPackageOwner
ICM_PRIV_ITEM_ROUTE_START	Used to see if the user is authorized to start a process.	startProcess
ICM_PRIV_ITEM_ROUTE_END	Used to see if the user is authorized to terminate a process.	terminateProcess
ICM_PRIV_ITEM_GET_WORKLIST	Used to see if the user is authorized to get the count or work packages from a worklist.	getCount listWorkPackagePidStrings
ICM_PRIV_ITEM_GET_WORK	Used to see if the user is authorized to get a work package.	getNextWorkPackagePidString getNextWorkPackage checkOutItemInWorkPackage retrieveWorkPackage
ICM_PRIV_ITEM_GET_ASGN_WORK	Used to see if the user is authorized to get a work package that is owned by a different a different user.	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings
ICM_PRIV_ITEM_ROUTE	Used to see if the user is authorized to route a work package.	continueProcess

Working with access control lists for document routing

When an ACL is defined for a document routing entity such as a process, work node, and worklist, the operations allowed on the entity are impacted. The effect of ACLs on DB2 Content Manager document routing entity and their associated privileges are listed in the following table.

Table 23. Access control lists and document routing

Objects	Related methods	Privileges
Process	startProcess	ICM_PRIV_ITEM_ROUTE_START
Work node	continueProcess suspendProcess resumeProcess terminateProcess setWorkPackagePriority setWorkPackageOwner	ICM_PRIV_ITEM_ROUTE ICM_PRIV_ITEM_ROUTE_END ICM_PRIV_ITEM_UPDATE_WORK
Worklist	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings checkOutItemInWorkPackage	ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST

Programming document routing user exits

A document routing user exit is a custom programming application (DLL file) that you can create specifically for a work node. You can set a work node to call a specific function in a specific DLL file under the following situations:

- Whenever a work package is created and started on a process.
- Whenever a work package moves to the work node.
- Whenever a work package leaves the work node.
- Whenever a collection point reaches a specific overload limit.

Whenever DB2 Content Manager calls a user exit, you can retrieve the work package from the work package table using the ComponentID in the myExit API.

To pass work package data (including container data) to and from the user exit, use the following ICMUSERSTRUCT (in this example, included in the WXV2UserExitSample.h):

WXV2UserExitSample.h

```
typedef struct ICMCONTAINERDATA_STRUCT
{
    char    szContainerName[33];
    char    szContainerVal[255];
} ICMCONTAINERDATA_STRUCT;

typedef struct ICMUSERSTRUCT
{
    long    lUserEvent;
    char    szWPCompID[19];
    char    szWPItemID[27];
    short   sWPVersionID;
    char    szRouteSel[33];
    short   sUpdateFlag;
    short   sNumContainerData; /* no. of icmcontainer data structs*/
    struct ICMCONTAINERDATA_STRUCT **ppContainerDataStruct;
} ICMUSERSTRUCT;
```

You can include this header when compiling your own function into a DLL file—for example, WXV2UserExitSample.dll. The following C example includes the header file and defines a custom function called WXV2UserExitSample.

The first part of WXV2UserExitSample lists the work package's current number of container data variables, its component ID, and its item ID into a text file IBMCMROOT/CM83_DR_User_Exit.txt.

WXV2UserExitSample.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "WXV2UserExitSample.h"
#if defined (WIN32)
    #include <windows.h>
    #include <process.h>
#elif defined (AIX) || defined (Solaris)
    #include <dlfcn.h>
#endif
extern long WXV2UserExitSample (ICMUSERSTRUCT *pCMStruct) {
#ifdef MVS
    char envStr[256];
    int i;
    ICMCONTAINERDATA_STRUCT * pContainerDataStruct;
    FILE *_file;
    #if defined (WIN32)
        strcpy(envStr, getenv("IBMCMROOT"));
        strcat(envStr, "\\CM83_DR_User_Exit_Sample_Output.txt");
    #elif defined (AIX) || defined (Solaris)
        strcpy(envStr, getenv("IBMCMROOT"));
        strcat(envStr, "/CM83_DR_User_Exit_Sample_Output.txt");
    #endif
    _file = fopen(envStr, "aw");
    fprintf(_file, "CM83 Document Routing user exit sample.\n");
    fprintf(_file, "The number of container data structures
        passed to this user exit: %d \n", pCMStruct->sNumContainerData);
    fprintf(_file, "The work package component ID passed to
        this user exit: %s \n", pCMStruct->szWPCompID );
    fprintf(_file, "The work package Item ID passed to this
        user exit: %s \n", pCMStruct->szWPItemID );
    #else
        printf("CM83 Document Routing user exit sample.\n");
        printf("The number of container data structures passed to
            this user exit: %d \n", pCMStruct->sNumContainerData);
        printf("The work package component ID passed to this
            user exit: %s \n", pCMStruct->szWPCompID );
        printf("The work package Item ID passed to this user
            exit: %s \n", pCMStruct->szWPItemID );
    #endif
}
```

Next, WXV2UserExitSample updates three work package container variables by setting sUpdateFlag to 1 (true), specifying the number of container variables in sNumContainerData, specifying each variable name in szContainerName, and specifying each variable value in szContainerVal.

WXV2UserExitSample.c

```
/*Create the container data to be returned to the LS. */
/* On return the LS will update the work package container data.*/
pCMStruct->sUpdateFlag = 1; /* update the container data */
pCMStruct->sNumContainerData=3; /* no. cont data structs returned.*/
pContainerDataStruct = (ICMCONTAINERDATA_STRUCT *)
malloc(sizeof(ICMCONTAINERDATA_STRUCT)* pCMStruct->sNumContainerData);
#ifdef MVS
    fprintf(_file, "Returning container data structures to LS.
    Number of container data structures being returned: %d \n",
    pCMStruct->sNumContainerData );
#else
    printf("Returning container data structures to LS. Number of
    container data structures being returned: %d \n",
    pCMStruct->sNumContainerData );
#endif
strcpy(pContainerDataStruct[0].szContainerName, "Loan amount");
strcpy(pContainerDataStruct[0].szContainerVal, "1000");
strcpy(pContainerDataStruct[1].szContainerName, "First name");
strcpy(pContainerDataStruct[1].szContainerVal, "Carly");
strcpy(pContainerDataStruct[2].szContainerName, "Last name");
strcpy(pContainerDataStruct[2].szContainerVal, "Morreale");
```

Then, WXV2UserExitSample sets which route that the work package should follow. In this example, szRouteSel is set to the Reject route.

WXV2UserExitSample.c

```
strcpy(pCMStruct->szRouteSel, "Reject"); /* On return from
this user exit, the LS will send the work package on this route.*/
#ifdef MVS
    for (i = 0; i < pCMStruct->sNumContainerData; i++)
    {
        fprintf(_file, "Container data name at %d is: %s \n", i,
        pContainerDataStruct[i].szContainerName );
        fprintf(_file, "Container data value at %d is: %s \n", i,
        pContainerDataStruct[i].szContainerVal );
    }
/* Updated the route, the LS will send the wp on this route.*/
fprintf(_file, "Returning the route to the LS, the work package
should take this route: %s \n", pCMStruct->szRouteSel);
fclose(_file);
#else
    for (i = 0; i < pCMStruct->sNumContainerData; i++)
    {
        printf("Container data name at %d is: %s \n", i,
        pContainerDataStruct[i].szContainerName );
        printf("Container data value at %d is: %s \n", i,
        pContainerDataStruct[i].szContainerVal );
    }
/* Update the route the work package is to take on return
from this user exit. */
printf("Returning the route to the LS, the work package
should take this route: %s \n", pCMStruct->szRouteSel);
#endif
```

Finally, WXV2UserExitSample updates the container variables and logs the process in IBMCMROOT/CM83_DR_User_Exit.txt.

WXV2UserExitSample.c

```
pCMStruct->ppContainerDataStruct=(ICMCONTAINERDATA_STRUCT **)
malloc(sizeof(ICMCONTAINERDATA_STRUCT *) *
pCMStruct->sNumContainerData);
for (i = 0; i < pCMStruct->sNumContainerData; i++)
{
    pCMStruct->ppContainerDataStruct[i] = &pContainerDataStruct[i];
}
return 0;
}
```

Because all work node user exits are only supported as a synchronous process (as a shared library), you should try to minimize the application's run time. Otherwise, the library server times out if the exit's running transaction takes too long.

Alternatively, to run the user exit as an asynchronous process, you can program the exit DLL to spawn a process that continues the business transaction separately from the library server.

Document routing constants

You define document routing constants in `DKConstantICM`. `DKConstantICM` contains the following document routing constants:

Worklist filtering parameters:

- `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;`

Workflow container variable data types:

- `public final static short DK_ICM_DR_WNV_TYPE_CHARACTER = 0;`
- `public final static short DK_ICM_DR_WNV_TYPE_INTEGER = 1;`
- `public final static short DK_ICM_DR_WNV_TYPE_TIMESTAMP = 2;`

Work node types:

- `public final static short DK_ICM_DR_WB_NODE_TYPE = 0;`
- `public final static short DK_ICM_DR_CP_NODE_TYPE = 1;`
- `public final static short DK_ICM_DR_SPLIT_NODE_TYPE = 2;`
- `public final static short DK_ICM_DR_JOIN_NODE_TYPE = 3;`
- `public final static short DK_ICM_DR_DP_NODE_TYPE = 4;`
- `public final static short DK_ICM_DR_SUB_PROCESS_NODE_TYPE = 5;`
- `public final static short DK_ICM_DR_BA_NODE_TYPE = 6;`

Process states:

- `public final static short DK_ICM_DR_PROCESS_VERIFIED_STATE = 0;`
- `public final static short DK_ICM_DR_PROCESS_DRAFT_STATE = 1;`

Chapter 7. Understanding prefetching in DB2 Content Manager for z/OS

The Prefetch feature enables you to move objects that are currently located on slower media, like optical or tape, to faster media, like direct access storage devices (DASD), or vice versa. This makes objects readily available for users to access. For example, XYZ Insurance has just been informed that they will be audited, and that they must provide records for the past seven years. The company policy is to migrate policies belonging to customers that have gone to other insurance companies to optical tape. In this scenario, the DB2 Content Manager system administrator must prefetch the policies of previous customers, and restore those policies to DASD. The policies can then be retrieved quickly in case an auditor wants to see them.

Prefetching objects

To use the prefetch feature, you call the DKLobICM and DKDatastoreICM APIs within your application. When calling the APIs, you specify that you want a particular item, or in the case of batch operations, a collection of items, to be pre-fetched. The APIs send a request to the resource manager to pre-fetch the object from the collection it resides in and place it in the collection that you specified as the pre-fetch collection at define time. Figure 13 on page 254 depicts the flow of a prefetch transaction:

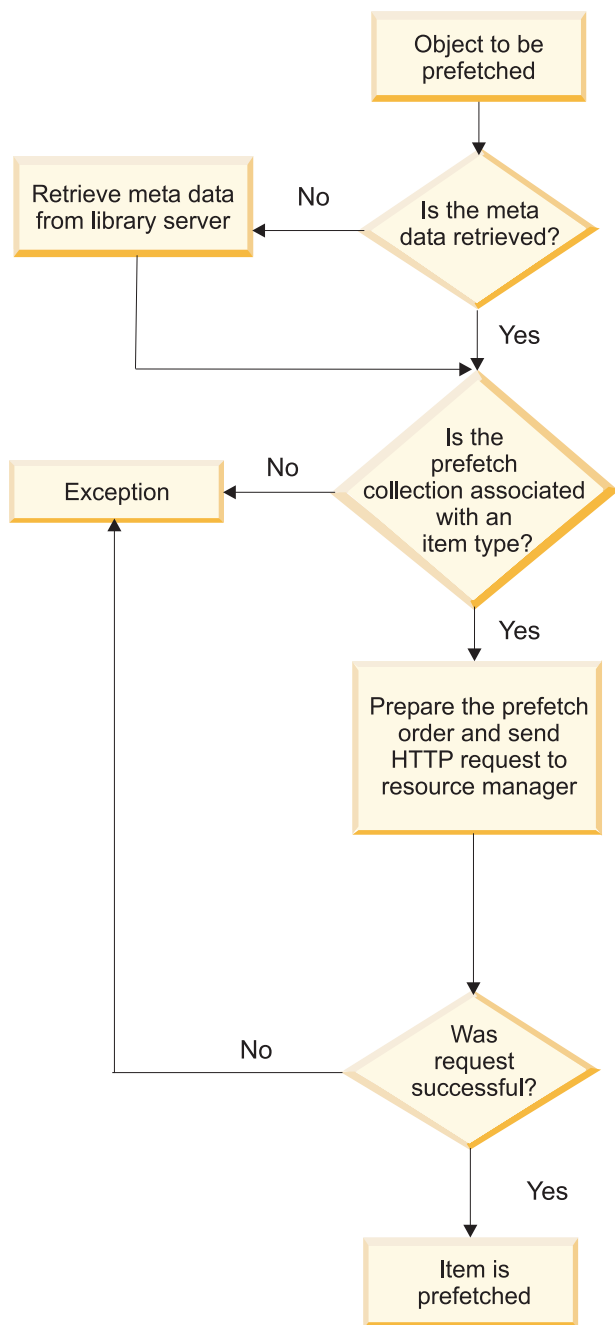


Figure 13. Flow of a prefetch transaction

Prefetching from the resource manager is an asynchronous transaction. Therefore, the successful completion of a prefetch transaction consists of the two parts described below.

1. The application calls the resource manager using a prefetch order. The prefetch order is then processed by the resource manager, inserting an entry into the ICMRMPREFETCH table. If this table insert is successful, a 0 return code is sent back to the calling application. At this stage, the object only exists in its original location or source collection. The table update process is as follows:
 - a. An entry is inserted into the ICMRMPREFETCH table.
 - b. The requesttimestamp and statetimestamp are set to the current timestamp.
 - c. The prefetchstate is set to INITIATED.

1. The actual processing, which is copying an object from the source to the prefetch collection, is done by the ICMMSAP asynchronous process. This process performs a select on the ICMRMCONTROL table for PREFETCHENABLED. If this value is set to 1, ICMMSAP processes the entries in the ICMRMPREFETCH table if any exist. This process is as follows:
 - a. The prefetchstate is updated to WORKING and the statetimestamp is updated with the current timestamp.
 - b. An OAM_Query is performed to see if the object exists within OAM by searching on its extobjname and source collection name.
 - c. If the object exists within OAM, an OAM_Retrieve operation is performed.
 - d. Once retrieved, an OAM_Store, for the existing extobjname, within OAM at the prefetch collection name, is performed.
 - e. If the OAM_Store is successful, the prefetchstate is updated to COMPLETE and the statetimestamp is updated with the current timestamp.
 - f. If a failure occurs, oamreturncode and oamreasoncode are updated with error codes from OAM, and prefetchstate is updated to FAILED.

Support for the prefetch APIs is limited to the Java APIs only. The APIs below are the APIs you work with to prefetch an object:

- DKDatastoreICM
`public dkCollection prefetchObjects(dkCollection prefetchColl,DKNVPair[] nvPairs) throws DKException,Exception`
- DKLobICM
`public boolean prefetchContent(DKNVPair[] nvPairs) throws DKException, java.lang.Exception`
`public boolean prefetchContent(DKNVPair[] nvPairs) throws DKException, java.lang.Exception`

For more specific information about the APIs, see the Application Programming Reference.

Table definitions related to prefetching

Table 24. ICMRMCONTROL

PREFETCHENABLED	SMALLINT	NOT NULL,
-----------------	----------	-----------

Table 25. ICMRMPREFETCH

ITEMID	CHAR(26)	NOT NULL
VERSIONID	SMALLINT	
VERSIONID	SMALLINT	
EXTOBJNAME	CHAR(44)	NOT NULL
SOURCECOLLNAME	CHAR(44)	NOT NULL
PREFETCHCOLL	NAME CHAR(44)	NOT NULL
REQUESTTIMESTAMP	TIMESTAMP	NOT NULL
STATETIMESTAMP	TIMESTAMP	NOT NULL
OAMRETURNCODE	SMALLINT	
OAMREASONCODE	CHAR(8)	
VOLSER	CHAR(6)	
OPERATION	VARCHAR(128)	NOT NULL
PREFETCHSTATE	VARCHAR(128)	NOT NULL

Table 25. **ICMRMPREFETCH** (continued)

ITEMID	CHAR(26)	NOT NULL
PRIMARY KEY (EXTOBJNAME)		

ITEMID

Generated by the library server during the object store transaction.
Example ITEMID: A1001001A03L09B64813I92553.

VERSIONID

When versioning is enabled, this value denotes a given instance of an object. Example VERSIONID: 1

EXTOBJNAME

The EXTOBJNAME is the ITEMID and VERSIONID combined and "." delimited, and represents a unique DB2 Content Manager object locator within Object Access Method (OAM). Example EXTOBJNAME: A1001001.A03L09.B64813.I92553.V001

SOURCECOLLNAME

The name of the collection that the object was originally stored under and, the source of the prefetch operation. Example SOURCECOLLNAME: CLLCT001

PREFETCHCOLLNAME

Generally represents a collection backed by a fast access medium such as DASD and is the target of a prefetch operation. Example PREFETCHCOLLNAME: CLLCT002

REQUESTTIMESTAMP

The time when a request was made to prefetch an object. Example REQUESTTIMESTAMP: 2003-12-09 21:48:22.778167000

STATETIMESTAMP

Reflects the time the PREFETCHSTATE was modified. For example, when an object is initially requested to be prefetched, the REQUESTTIMESTAMP and the STATETIMESTAMP are the same and the prefetchstate is INITIATED. Once the processing of the prefetch request begins, the PREFETCHSTATE is updated to WORKING. The STATETIMESTAMP is also updated to reflect the time the prefetch processing began. If the prefetch processing completes successfully, the PREFETCHSTATE is updated to COMPLETE. If there is a failure, the PREFETCHSTATE is updated to FAILED. In either case, the STATETIMESTAMP is also updated with the current timestamp. Example STATETIMESTAMP: 2003-12-09 21:48:22.778167000

OAMRETURNCODE

An integer value that represents the return code from an OAM operation. Prefetching an object requires multiple calls to the OAM interface. Therefore, the return code from OAM indicates the success or failure of a given prefetch operation. Since the prefetch function currently runs asynchronously, the calling application has no way of knowing the status of a given prefetch operation, therefore it is important to store this status information persistently for later status query. Example OAMRETURNCODE: 8

OAMREASONCODE

Provides detailed information about the OAMRETURNCODE failure and is

| the character representation of the hexadecimal value used to locate the
| reason code description in the DFSMSdfp Diagnosis Reference Guide
| (GY27-7618-03). Example OAMREASONCODE: 2C040100

| **VOLSER (for future use)**

| Represents the tape volume on which a given object can be stored. This
| value is included for the purpose of grouping prefetch requests according
| to the tape volume where they reside to prevent “thrashing” of the tape
| drives.

| **OPERATION**

| This value indicates the operation that is to be performed. Currently, the
| only valid value for this column is PREFETCH. The operation column is
| included for future extendibility and therefore will contain other values in
| the future. Example OPERATION: PREFETCH

| **PREFETCHSTATE**

| Indicates the progress of a prefetch transaction. The list below provides the
| valid values for this column:

- | • INITIATED: The initial state when a prefetch request is inserted into the
| ICMRMPREFETCH table.
- | • WORKING: The updated state indicating that the prefetch process has
| started processing the given object’s prefetch request.
- | • COMPLETE: The updated state indicating that the prefetch process has
| ended successfully.
- | • FAILED: The updated state indicating that the prefetch process returned
| with a non-zero error condition.

Chapter 8. Working with other content servers

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

To create a content server, use the `DKDatastorexx` classes, where `xx` identifies the specific content server. Table 5 on page 23 shows these classes.

Table 26. Server type and class name terminology

Content server	Class name
DB2 Content Manager Version 8.3	<code>DKDatastoreICM</code>
Earlier DB2 Content Manager	<code>DKDatastoreDL</code>
Content Manager OnDemand	<code>DKDatastoreOD</code>
DB2 Content Manager for AS/400 (VisualInfo for AS/400)	<code>DKDatastoreV4</code>
Content Manager ImagePlus for OS/390	<code>DKDatastoreIP</code>
Domino.Doc	<code>DKDatastoreDD</code>
Relational databases	<code>DKDatastoreDB2</code> , <code>DKDatastoreJDBC</code> (for Java), <code>DKDatastoreODBC</code>

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

dkDatastore

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

dkDatastoreDef

Uses the methods to access items stored in the content server. Also creates, lists, and deletes its entities. It maintains a collection of `dkEntityDefs`. Examples of concrete classes for this interface are:

- `DKDatastoreDefDL`
- `DKDatastoreDefOD`

dkEntityDef

Uses the methods to access entity information. Also creates and deletes entities and attributes. The methods of this class support accessing multiple-level entities. If a content server does not support subentities, they generate `DKUsageError` objects. If a content server supports multiple-level entities, you must implement methods to overwrite the exceptions for subclasses for these content servers. Examples of concrete classes for the `dkEntityDef` interface are:

- `DKIndexClassDefDL`
- `DKAppGrpDefOD`

The class hierarchy for an entity definition is illustrated in Figure 14:

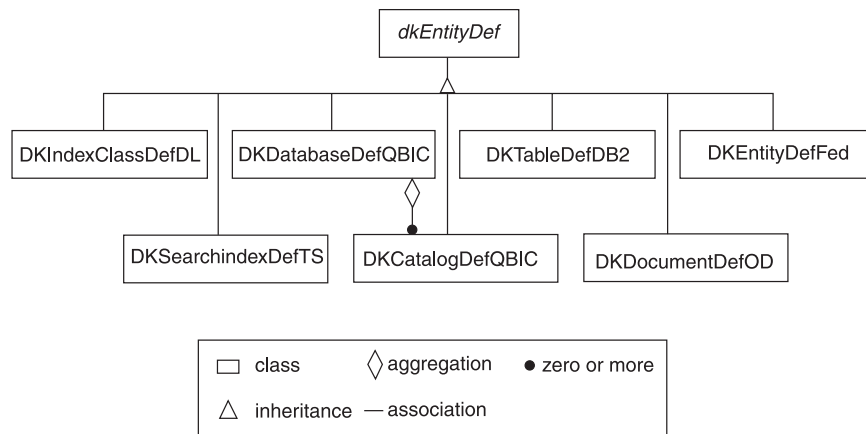


Figure 14. Class hierarchy

dkAttrDef

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

- DKAttributeDefDL
- DKFieldDefOD

dkServerDef

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

- DKServerDefDL
- DKServerDefOD

dkResultSetCursor

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

dkBlob

Declares a common public interface for binary large object (BLOB) data types in each content server. The concrete classes derived from dkBlob share this common interface, allowing processing of BLOBs from heterogeneous content servers. Examples of concrete classes for dkBlob are:

- DKBlobDL
- DKBlobOD

The data definition classes and their class hierarchy are represented in Figure 15 on page 261:

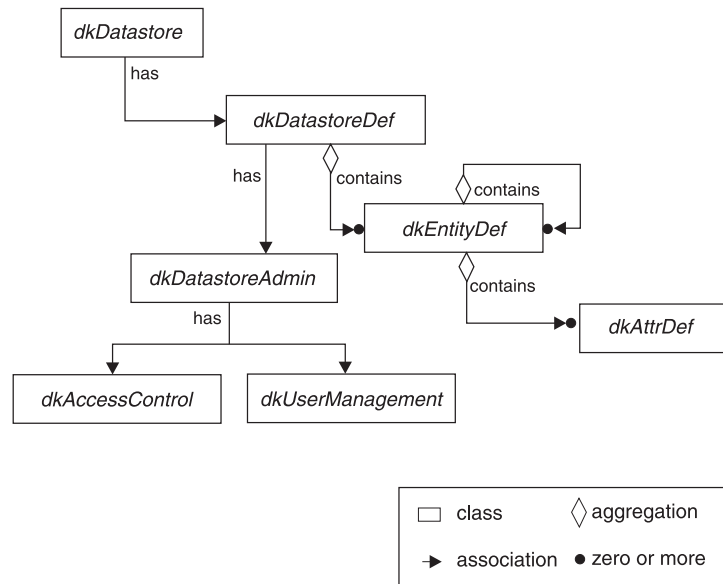


Figure 15. Data definition class hierarchy

For more information on `dkDatastore` and other common classes, see “Developing custom content server connectors” on page 368.

Working with earlier DB2 Content Manager

This section describes how to access data in DB2 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 DB2 Text Information Extender.
- Use image search (QBIC®).
- Use workflows and workbaskets.

Handling large objects

In the earlier DB2 Content Manager connector, you can retrieve large objects piece by piece using asynchronous retrieval. For the sample application, see `TxdoAsyncRetDL` in the Samples directory.

Setting Java heap size (Java only)

Java has a limitation for the default initial and maximum heap size. The default initial heap size is 1 048 576 and the default maximum heap size is 16 777 216 bytes. If your Java application program tries to use objects larger than the heap size, your program will fail during execution. To increase maximum heap size for your application, use the `-mx` option when you execute your Java application program. For example:

Java

```
java -mx40000000 yourApplication
```

Using DDOs to represent earlier Content Manager content

A DDO associated with DKDatastoreDL has some specific information to represent the Information Integrator for Content document model: document, folder, parts, item, item ID, rank, and so forth. The following sections describe how you access this information.

DDO properties

The type of an item, whether it is a document or folder, is a property under the name DK_CM_PROPERTY_ITEM_TYPE. To get the item type of the DDO, you call:

Java

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

C++

```
DKAny any = cddo->getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
if (!any.isNull()) {
    unsigned short item_type = (unsigned short) any;
}    ...        // do something
```

After the property is called, the `item_type` is equal to `DK_CM_DOCUMENT` for a document, or `DK_CM_FOLDER` for a folder. The if statement ensures that the property exists. See “Adding properties to a DDO” on page 30 and “Getting the DKDDO and attribute properties” on page 33 for more information.

Persistent identifier (PID)

The PID contains important pieces of information specific to Information Integrator for Content: the object type indicates the index class the DDO belongs to; the PID contains the item ID of the associated item from the content server. See “Creating a persistent identifier (PID)” on page 30.

Representing documents

A DDO representing a document has the property `DK_CM_PROPERTY_ITEM_TYPE` set to `DK_CM_DOCUMENT`. Its PID contains the index class name as the object type. The PID ID the same as the item ID.

The parts inside a document are represented as `DKPartsDL` objects, which are collections of binary large objects (BLOBs), each of which is represented as a `DKBlobDL` object.

A document DDO has a specific attribute named `DKPARTS`, whose value is a `DKParts` object.

To get to each part in a document, retrieve the `DKParts` first, then create an iterator to iterate over the parts. If the document does not have any parts, `DKParts` is null or the cardinality of `DKParts` is zero.

Documents associated with a combined query (a combination of a parametric and text query) can have a transient attribute named `DKRANK`, whose value is an object containing an integer rank computed by the DB2 Text Information Extender.

For more information on creating and processing a DKParts object, see “Creating, updating, and deleting documents or folders,” “Retrieving a document or folder” on page 271, and “Creating documents and using the DKPARTS attribute” on page 67.

Representing folders

A DDO representing a folder has a property `DK_CM_PROPERTY_ITEM_TYPE` equal to `DK_CM_FOLDER`. Similar to a document DDO, its PID contains the index class name as the object type, and item ID in the PID’s ID.

A DKFolder object represents the table of contents inside a folder. A DKFolder object is a collection of DDOs. Each DDO represents an item in the folder, either a document or another folder. A folder DDO has an attribute named `DKFOLDER`, whose value is a DKFolder object.

To get to each DDO member of the folder, retrieve the DKFolder object first; then create an iterator to access each item member. If the folder does not have a member, DKFolder is null, but the DKFOLDER attribute is always present in a folder DDO created by the content server.

For more information on creating and processing a DKFolder object, see “Creating, updating, and deleting documents or folders,” “Retrieving a document or folder” on page 271, and “Creating folders and using the DKFOLDER attribute” on page 70.

Creating, updating, and deleting documents or folders

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

Creating a document

To create a document and save its persistent data in a content server, you must create a DDO and set all of its attributes (and other information) except for the item ID. The item ID is assigned and returned by the content server. Some of the previous examples are combined in the following example:

Java

```
// ----- Step 1: create a datastore and connect to it
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Step 2: create a document (or folder) DDO
//         and set all its attributes and other required information
DKPid pid = new DKPid();
pid.setObjectType("GRANDPA"); // Set the index-class name it belongs to
DKDDO ddo = new DKDDO(dsDL,pid); // Create a DDO with PID and
...                               // associate it to dsDL

// ----- Step 2.a: add attributes according to index class GRANDPA
Object obj, vstr;
Boolean yes = new Boolean(true);
Boolean no = new Boolean(false);

short data_id = cddo.addData("Title"); // add new attribute "Title"
vstr = new Short(DK_CM_DATAITEM_TYPE_STRING);
// ----- Add type properties VSTRING and nullable
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, no);

data_id = cddo.addData("Subject"); // add new attribute "Subject"
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, yes);

// ----- Add some more attributes as necessary
....

// ----- Step 2.b: add DKPARTS attribute
DKParts parts = new DKParts(); // create a new DKParts, collection of parts
DKBlobDL blob = new DKBlobDL(dsDL); // create a new XDO blob
DKPidXDODL pidXDO = new DKPidXDODL(); // create PID for this XDO object

pidXDO.setPartId(5); // set part number to 5
blob.setPidObject(pidXDO); // set the PID for the XDO blob
blob.setContentClass(DK_DL_CC_GIF); // set content class type GIF
blob.setRepType(DK_REP_NULL); // set rep type for the part
blob.setContentFromClientFile("choice.gif"); // set the blob's content
blob.setInstanceOpenHandler("xv"); // the viewer program on AIX

parts.addElement(blob); // add the blob to the parts collection

.... // create and add some more blobs to
.... // to the collection as necessary

// ----- Create DKPARTS attribute and set it to refer to the DKParts object
short data_id = ddo.addData(DKPARTS); // add attribute "DKParts"
obj = new Short(DK_CM_COLLECTION_XDO); // add type property
ddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, obj);
ddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, yes); // add nullable prop
ddo.setData(data_id, parts); // sets the attribute value

// ----- Step 2.c: sets the item type : document
obj = new Short(DK_CM_DOCUMENT);
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, obj);

// ----- Step 3: make item persistent; add item to the datastore
ddo.add(); // document created in datastore
```

C++

```
// step 1: create a datastore and connect to it
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// step 2: create a document (or folder) DDO
//          and set all its attributes and other required information
//          See the section on "Using DDO"
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// create a DDO with PID and associated with dsDL
DKDDO* ddo = new DKDDO(&dsDL,pid);

// step 2.a: add attributes according to index-class GRANDPA
DKAny any;
DKBoolean yes = TRUE;
DKBoolean no = FALSE;
// add a new attribute named "Title"
unsigned short data_id = ddo->addData("Title");
// add type property VSTRING
any = DK_VSTRING;
ddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
// add nullable property: non-nullable
any = no;
ddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add a new attribute named "Subject"
data_id = ddo->addData("Subject");

any = DK_VSTRING;
ddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
any = yes;
ddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add some more attributes as necessary
// ...

// step 2.b: add DKPARTS attribute
// create a new XDO blob
DKParts* parts = new DKParts;
DKBlobDL* blob = new DKBlobDL(&dsDL);

DKPidXDODL pidXDO;                                // create PID for this XDO object

pidXDO.setPartId(5);                                // set part number to 5
blob->setPid(&pidXDO);                                // set the PID for the XDO blob
blob->setContentClass(DK_CC_GIF);                    // set content class type GIF
blob->setRepType(DK_REP_NULL);                        // set rep type for the part
blob->setContentFromClientFile("choice.gif"); // set the blob's content

DKAny any = (dkDataObjectBase*) blob;
parts->addElement(any);                                // add the blob to the parts collection

...                                                    // create and add some more blobs
...                                                    // to the collection as necessary
// continued...
```

C++ (continued)

```
// create DKPARTS attribute and sets it to refer to the DKParts object
// add attribute "DKParts"
unsigned short data_id = ddo->addData(DKPARTS);
any = DK_COLLECTION_XDO;
// add type property
ddo->addDataProperty(data_id,DK_CM_PROPERTY_TYPE,any);
any = (DKBoolean) TRUE;
// add nullable property
ddo->addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE ,any);
any = (dkCollection*) parts;
// sets the attribute value
ddo->setData(data_id, any);

// step 2.c: sets the item type : document
any = DK_CM_DOCUMENT;
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);

// step 3: make it persistent
// a document is created in the datastore
ddo->add();
```

The last step of the preceding example created a document in the content server (with the information). Whenever a document DDO is added to a content server, all of its attributes are added, including all of the parts inside the DKParts collection.

You use the same process for adding a folder DDO. The DKFOLDER collection members are added to the content server as a component of the folder. The folder contains a table of contents of its members, which are existing documents and folders. Therefore, create all folder members in the content server before adding a folder DDO.

Java

You can add the same document to a different content server of the same type. To add this document to the DB2 Content Manager server LIBSRVRN, which has an index class LIBSV2 with the same structure as LIBSV, use the following example:

```
// ----- Create datastore and connect to LIBSRVRN
DKDatastoreDL dsN = new DKDatastoreDL();
dsN.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");

// ----- Update the PID
pid = ddo.getPidObject();
pid.setObjectType("LIBSV2");           // set the new index class
pid.setPrimaryId("");                  // make the item ID blank
pid.setDatastoreName("LIBSRVRN");      // set the new datastore name
ddo.setPidObject(pid);                 // update the PID
ddo.setDatastore(dsN);                 // re-associate the DDO with dsN
ddo.add();                             // add the DDO
```

C++

You can add the same document to a different content server of the same type. For example, to add the document to the server LIBSRVRN, which has an index class GRANDPA2 with the same structure as GRANDPA:

```
// create datastore and connect to LIBSRVRN
DKDatastoreDL dsN;
dsN.connect("LIBSRVRN","FRNADMIN","PASSWORD");
// update the Pid
pid = ddo->getPid();
pid.setObjectType("GRANDPA2");           // set the new index-class
pid.setId("");                           // blank the item-id
pid.setDatastoreName("LIBSRVRN");        // set the new datastore name
ddo->setPid(pid);                         // update the PID
ddo->setDatastore(&dsN);                  // re-associate it with dsN
ddo->add();                               // add it
```

Updating a document or a folder

To update a document or folder:

1. Set the item ID and the object type.
2. Update the appropriate attributes, or add to the DKParts collection.
3. Call the update method to store the change.

Java

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

C++

```
// update the value of attribute Title
DKAny any = DKString("Guess who is behind all this");
unsigned short data_id = ddo->getDataByName("Title");
ddo->setData(data_id, any);
ddo->update();
```

After the call to the update method, the value of the attribute Title in the content server is updated. The parts in this document are not updated unless their content has changed. The connection to the server must be valid when you call the update method.

Update a folder DDO using similar steps: update the attribute values, or add or remove elements from DKFolder; then call the update method.

Updating parts

Represent the collection of parts in a document using a DKParts object.

DKParts is a subclass of DKSequentialCollection. In addition to inheriting the sequential collection functions, DKParts has two additional methods for adding a part to, and removing a part from, the collection. These methods also immediately save the changes to the content server.

The document must already exist in the content server.

Adding and removing a member: The following examples add a part to a document.

Java

```
DKDDO addo = new DKDDO();    // create a document DDO
DKBlobDL newPart = new DKBlobDL(); // create the new part to be added
....                          // initialized the DDO and new part
DKParts parts = (DKParts) addo.getDataByName(DKPARTS); // get DKParts
parts.addMember(ddo, newPart); // assume none of these values are NULL
```

C++

```
// a document DDO
DKDDO* ddo;
// a new part to be added
DKBlobDL* newPart;
// ddo and newPart are
// initialized somewhere along the line
...
...
// get DKParts
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
// assume none of these values are NULL
parts->addMember(ddo, newPart);
```

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

Java

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

C++

```
parts->removeMember(ddo, newPart);
```

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

Differences between update, add, and remove on a document DDO: The addMember and removeMember methods of DKParts provide conveniences for adding and removing a part in the collection and the content server. They are faster than the update method in a document DDO. The update method on a DDO updates all of the attributes in the DDO, including DKParts and all of its members that changed. The steps are:

Java

```
....
// ----- Get DKParts, assume it exists and not null
DKParts parts = (DKParts) addo.getDataByName(DKPARTS);
parts.addElement(newPart);           // add a new part to parts
addo.update();                       // updates the whole ddo
....
```

C++

```
...
DKAny any = ddo->getDataByName(DKPARTS);
// get DKParts, assume it exists
DKParts* parts = (DKParts*) any.value();
// assume it is not NULL
any = (dkDataObjectBase*) newpart;
parts->addElement(any);
// updates the whole ddo
ddo->update();
...
```

Updating folders

You represent the collection of documents and folders within a folder using a DKFolder object. In the content server, a folder holds a table of contents referring to its objects instead of keeping the actual objects.

DKFolder is a subclass of DKSequentialCollection. In addition to inheriting the sequential collection methods, it has two additional members for adding a member (a document or a folder) to, or removing a member from, the collection and immediately stores those changes.

The document or folder to be added or removed must already exist in the content server.

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

Java

```
DKDDO folderDDO = new DKDDO(); // Created the folder DDO
DKDDO newMember = new DKDDO(); // Create the new DDO to be added
....                          // The folder DDO and newMember are
....                          // initialized
// ----- Get the DKFolder, assuming it exists, and the value not null
DKFolder folder = (DKFolder) folderDDO.getDataByName(DKFOLDER);
folder.addMember(folderDDO, newMember);
```

C++

```
// a folder DDO
DKDDO* folderDDO;
// a new DDO to be added as a member of this folder
DKDDO* newMember;
...      // folderDDO and newMember are
...      // initialized somewhere along the line
DKAny any = folderDDO->getDataByName(DKFOLDER);
// get DKFolder, assume it exists
DKFolder* folder = (DKFolder*) any.value();
// assume non NULL
folder->addMember(folderDDO, newMember);
```

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

Similarly, to remove `newMember` from the collection and the content server use the following example:

Java

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

C++

```
folder->removeMember(folderDDO, newMember);
```

Important: Removing a member from a folder only removes that member from the folder table of contents. If you use the `removeElementAt`, then function it does not delete the member from memory or from the content server.

Differences between update, add, and remove on a folder DDO: The `addMember` and `removeMember` methods of `DKFolder` provide conveniences for adding and removing a document or folder in the collection and in the content server. They are faster than the update method in a folder DDO

The update method on a DDO updates all of the attributes in the DDO, including `DKFolder` and all of its members, whereas the `addMember` and `removeMember` methods only add or remove a member in the folder table of contents.

Deleting a document or a folder

Use the `del` method in the DDO to delete a document or folder from the content server.

Java

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

C++

```
ddo->del();
```

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

Use the statement above to delete a folder as well. Only persistent data is deleted, the in-memory copy of the DDO does not change. Therefore, you can add this DDO back to the same or different content server later, in the application. See “Creating a document” on page 263 for more information.

Retrieving a document or folder

To retrieve a document from a DKDatastoreDL (representing an earlier DB2 Content Manager content server), you must know the document’s index class name and item ID. You also must associate the DDO with a content server and establish a connection.

Java

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

DKPid pid = new DKPid();
pid.setObjectType("Claim"); // set the index-class name it belongs to
pid.setPrimaryId("LN#U5K6ARLGM3DB4"); // set the item-id
// ----- create a DDO with the PID and associated with the datastore

ddo.retrieve(); // retrieve the document
```

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// set the item-id
pid.setId("LN#U5K6ARLGM3DB4");
// create a DDO with pid and associated to dsDL
DKDDO* ddo = new DKDDO(&dsDL, pid);
// retrieve it
ddo->retrieve();
```

After a call to retrieve, all of the DDO’s attribute values are set to the value of the persistent data stored in the content server. If the document has parts, the DKPARTS attribute is set to a DKParts object. However, the content of each part in this collection is not retrieved. Because a part might be large, you should not retrieve all of them into memory at once. It is better to explicitly retrieve the part you want.

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

Retrieving parts

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

Java

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

C++

```
DKAny any = ddo->getDataByName(DKPARTS);  
DKParts* parts = (DKParts*) any.value();
```

This example assumes that the DKPARTS attribute exists. If it does not exist, an exception is generated. See “Retrieving a folder” on page 273 for an example of extracting an attribute value by getting the data ID first and testing it for zero.

To retrieve each part, you must create an iterator to step through the collection and retrieve each part. See “Creating documents and using the DKPARTS attribute” on page 67.

Java

```
// ----- Create an iterator and process the part collection members  
if (parts != null) {  
    DKBlobDL blob;  
    dkIterator iter = parts.createIterator();  
    while (iter.more()) {  
        blob = (DKBlobDL) iter.next();  
        if (blob != null) {  
            blob.retrieve(); // retrieve the blob's content  
            blob.open();  
            .... // other processing, as needed  
        }  
    }  
}
```

C++

```
// create iterator and process the part collection member one by one  
if (parts != NULL) {  
    DKAny* element;  
    DKBlobDL* blob;  
    dkIterator* iter = parts->createIterator();  
    while (iter->more()) {  
        element = iter->next();  
        blob = (DKBlobDL*) element->value();  
        if (blob != NULL) {  
            // retrieve the blob's content  
            blob->retrieve();  
            // other processing, as needed  
            blob->open();  
        }  
    }  
    delete iter;  
}
```

Similar to the DDO results of a parametric query, each part XDO inside the DKParts collection is empty (does not have its content set). However, it has all the information needed for retrieval. To bring its content and related information into memory, call the retrieve method:

Java

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

C++

```
blob->retrieve();
```

Retrieving a folder

Retrieve a folder DDO in the same way that you retrieve a document DDO. After being retrieved, the folder DDO has all of its attributes set, including the attribute, DKFOLDER. This attribute value is set to a DKFolder object, a collection of the DDO members in the folder. Like the parts in a DKParts object, these member DDOs contain only enough information to retrieve them. You can retrieve a folder DDO as follows:

Java

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

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

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

C++

```
// get DKFOLDER data-id
data_id = ddo->dataId(DKFOLDER);
// folder not found
if (data_id == 0) {
    DKException exc(" folder data-item not found");
    DKTHROW exc;
}
// get the folder collection
any = ddo->getData(data_id);
DKFolder* fCol = (DKFolder*) any.value();
// create iterator and process the DDO collection member one by one
if (fCol != NULL) {
    DKAny* element;
    DKDDO* item;
    dkIterator* iter = fCol->createIterator();
    while (iter->more()) {
        element = iter->next();
        item = (DKDDO*) element->value();
        if (item != NULL) {
            // retrieve the member DDO
            item->retrieve();
            // other processing
            ...
        }
    }
    delete iter;
}
```

See also “Creating folders and using the DKFOLDER attribute” on page 70.

Understanding text searching (DB2 Text Information Extender)

The DB2 Text Information Extender product supports various query types:

- “Boolean query” on page 275
- “Free text query” on page 275
- “Hybrid query” on page 275
- “Proximity query” on page 276
- “Global text retrieval (GTR) query” on page 276

You can use the text search item ID, part number, and ranking information (returned by the query) to create an XDO that retrieves the document from an earlier DB2 Content Manager server.

Use a DKDatastoreTS object to represent the DB2 Text Information Extender. DB2 Text Information Extender does not actually store the data, it merely indexes the data stored in earlier DB2 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 DB2 Content Manager. Use these identifiers to retrieve the document.

The DKDatastoreTS object does not support add, update, retrieve, and delete functions. You can query this content server. See “Loading data to be indexed by DB2 Text Information Extender” on page 284 for information on adding data to DB2 Content Manager that is indexed by DB2 Text Information Extender.

Boolean query

A boolean query is made up of words and phrases, separated by boolean operators. Enclose a phrase in single quotes ('). Phrases are treated as a literal strings.

The following example creates a query string to search for all text documents with the word `www` or the phrase `web site` in the `TMINDEX` text search index:

Java

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

C++

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

Free text query

A free text query is made up of words, phrases, or sentences enclosed in braces ({ }). The words are not required to be adjacent to each other. The following example creates a query string to search for all text documents with the free text `web site` in the `TMINDEX` text search index:

Java

```
String cmd = "SEARCH=(COND=({web site}));" +  
            "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=({Web site}))";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Hybrid query

A hybrid query is made up of a boolean query followed by a free text query. The following example creates a query string to search for all text documents with the words `IBM` and `UNIX`, as well as the free text `web site` in the `TMINDEX` text search index:

Java

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

C++

```
DKString cmd = "SEARCH=(COND=(IBM AND UNIX {Web site}))";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```


Proximity query

A proximity query looks for a sequence of search arguments found in the same document, paragraph, or sentence. The following example creates a query string to search for all text documents with the phrase rational numbers and the word math in the same paragraph using the TMINDEX text search index:

Java

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

C++

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

This type of query requires at least two search arguments.

Global text retrieval (GTR) query

A GTR query is optimized for double-byte character set (DBCS) languages like Japanese or Chinese. GTR also supports single-byte character set (SBCS) languages. Enclose all double-byte characters in single quotes ('). Make sure that the phrase to be searched for is in the specified character code set and language.

The following example shows a GTR search for all text documents that contain the phrase IBM marketing. The MATCH keyword is set to indicate the degree of similarity for the phrase.

Java

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

C++

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

Make sure that the text search content server options DK_OPT_TS_CCSID (coded character set identifiers) and DK_OPT_TS_LANG (language identifiers) are set properly. The default for DK_OPT_TS_CCSID is DK_CCSID_00850. The default for DK_OPT_TS_LANG is DK_LANG_ENU. These values are used as the global defaults for the text query. For more information, see the Application Programming 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 DB2 Text Information Extender information using DDOs

You use a DDO (associated with a DKDatastoreTS object) to represent the results from text searches.

DKDastastoreTS does not have a property item type as a DKDatastoreDL object does. The format of its ID is also different. A DDO resulting from a text query corresponds to a text part inside an item. It contains the following standard attributes:

DKDLITEMID

The item ID that this text is part of. Use this item ID to retrieve the whole item from the content server.

DKPARTNO

An integer part number for this text part. Use the part number with the item ID to retrieve the text part from the content server.

DKREPTYPE

The RepType of this text part. Use this attribute with the item ID and part number to retrieve the text part from the content server.

DKRANK

An integer rank signifying the relevance of this part to the results of a text query. A higher rank means a better match. See the Application Programming Reference for further information.

DKDSIZE

An integer number representing word occurrences (in the results of boolean queries). See the Application Programming Reference for further information.

DKRCNT

An integer number representing boolean search matches. See the Application Programming Reference for further information.

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

content server type

TS.

content server name

The name used to connect to the content server.

object type

DB2 Text Search Extender index.

ID DB2 Text Information Extender document ID.

Establishing a connection

The DKDatastoreTS object provides two functions for connecting and a function for disconnecting. Normally, you create a DKDatastoreTS object, connect to it, run a query, and then disconnect when done. The following example shows the first connection function using the text search server .

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("TM", "", "", "");
.... // run a query
dsTS.disconnect();
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","", "", "");
... // do some work
dsTS.disconnect();
```

The following example shows the second connection function using the text search server with the hostname `apollo`, port number 7502, and TCP/IP communication type `DK_CTYP_TCPIP`:

```
dsTS.connect("apollo", "7502", DK_CTYP_TCPIP);
```

The following example shows the first connection function using the text search server hostname `apollo`, port number 7502, communication type T (TCP/IP):

```
dsTS.connect("apollo", "", "", "PORT=7502; COMMTYPE=T");
```

The following example shows the first connect method using the text search server name `TM` and using library server `LIBSRVR2`, user ID `FRNADMIN` and password `PASSWORD`:

The following example shows the first connection function using the text search server name `TM`, library server `LIBSRVRN`, user ID `FRNADMIN`, and password `PASSWORD`:

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

You can use the last parameter `LIBACCESS`, also called the connect string, to pass a sequence of parameters.

Tip: To prevent the DB2 Text Information Extender connection from timing out, connect to DB2 Text Information Extender, run your queries, and immediately disconnect. Do not leave the connection open.

Getting and setting text search options

DB2 Text Search Extender provides some options that you can set or get using its functions. See the Application Programming Reference for the list of options and their descriptions. The following example shows how to set and get the option for a text search character code set.

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
Integer input_option = new Integer(DK_TS_CC SID_00850);
Integer output_option = null;

dsTS.setOption(DK_TS_OPT_CC SID, input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CC SID);
```

C++

```
DKAny input_option = DK_CCSID_00850;
DKAny output_option;
dsTS.setOption(DK_OPT_TS_CCSID,input_option);
dsTS.getOption(DK_OPT_TS_CCSID,output_option);
```

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

Tips: The search options CCSID and LANG go together. If one is specified, the other must also be specified. The default CCSID and LANG are specified by the DKDatastoreTS options, DK_OPT_TS_CCSID and DK_OPT_TS_LANG. See the Application Programming Reference for the list of the content server options and their descriptions.

You can specify more than one search option for a query term. The search options are separated by commas. An example of multiple search terms is given in “Global text retrieval (GTR) query” on page 276.

If both the SC (single required character) and the MC (sequence of optional characters) search options, you must specify the SC search option first. For example, \$SC=?,MC=*\$ U?I*.

Listing servers

The DKDatastoreTS object provides a function to list the text search servers that it can connect to. The following example shows how to retrieve the list of servers.

Java

```
DKServerDefTS pSV = null;
DKIndexTS pIndx = null;
String strServerName = null;
char chServerLocation = ' ';
String strLoc = null;
String strIndexName = null;
String strLibId = null;
int i = 0;
DKDatastoreTS dsTS = new DKDatastoreTS();
System.out.println("list servers");
pCol = (DKSequentialCollection)dsTS.listDataSources();
pIter = pCol.createIterator();
while (pIter.more() == true)
{
    i++;
    pSV = (DKServerDefTS)pIter.next();
    strServerName = pSV.getName();
    chServerLocation = pSV.getServerLocation();
    if (chServerLocation == DK_TS_SRV_LOCAL)
        strLoc = "LOCAL SERVER";
    else if (chServerLocation == DK_TS_SRV_REMOTE)
        strLoc = "REMOTE SERVER";
    System.out.println("Server Name [" + i + "] - " + strServerName +
        " Server Location - " + strLoc);
}
```

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

C++

```
DKDatastoreTS dsTS;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKString strServerName;
char chServerLocation = ' ';
DKString strLoc;
DKServerDefTS *pSV = 0;
long i = 0;
DKAny a;
cout << "list servers" << endl;
a = dsTS.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
    i++;
    pSV = (DKServerDefTS*)((void*)(*pIter->next()));
    strServerName = pSV->getName();
    chServerLocation = pSV->getServerLocation();
    if (chServerLocation == DK_SRV_LOCAL)
    {
        strLoc = "LOCAL SERVER";
    }
    else if (chServerLocation == DK_SRV_REMOTE)
    {
        strLoc = "REMOTE SERVER";
    }
    cout << "Server Name [" << i << "] - " << strServerName
        << " Server Location - " << strLoc << endl;
    delete pSV;
}
delete pIter;
delete pCol;
```

The complete sample application from which this example was taken (TListCatalogTS.cpp) is available in the directory.

The list of servers is returned in a DKSequentialCollection of DKServerInfoTS objects. After you get a DKServerInfoTS object, you can retrieve the server name and location. You can then use the server name to establish a connection to it.

Listing schema

A DKDatastoreTS object provides functions for listing the schema. For text search, these are text search indexes. The following example shows how to retrieve the index list.

The list of indexes is returned in a DKSequentialCollection object of DKIndexTS objects. After you get a DKIndexTS object, you can retrieve information about the index, such as its name and library ID, which you can use to form a query.

Java

```
tsCol = (DKSequentialCollection) dsTS.listEntities();
tsIter = pCol.createIterator();
int i = 0;
while (tsIter.more()) {
    i++;
    TsIndx = (DKSearchIndexDefTS)tsIter.next();
    strIndexName = TsIndx.getName();
    strLibId = TsIndx.getLibraryId();
    ...           \\ Process the list as appropriate
}
```

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

C++

```
DKDatastoreTS dsTS;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKString strIndexName;
DKString strLibId;
DKServerDefTS *pSV = 0;
DKSearchIndexDefTS *pIndx = 0;
long i = 0;
DKAny a;
cout << "connecting to datastore" << endl;
dsTS.connect("TM","","");
cout << "list search indexes" << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsTS.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
    i++;
    pIndx = (DKSearchIndexDefTS*)((void*)(*pIter->next()));
    strIndexName = pIndx->getName();
    strLibId = pIndx->getLibraryId();
    cout << "index name [" << i << "] - " << strIndexName
        << " Library - " << strLibId << endl;
    delete pIndx;
}
delete pIter;
delete pCol;
dsTS.disconnect();
```

The complete sample application from which this example was taken (TListCatalogTS.cpp) is available in the samples directory. Also, refer to "Managing memory in collections (C++ only)" on page 78 for information about deleting the collection.

Indexing XDOs by search engine

Before searching data items with the DB2 Text Information Extender and image search, you must first index the data. Indexes require three values: SearchEngine, SearchIndex and SearchInfo.

The value of the SearchIndex property is a combination of two names: the search service name and search index name. For example, if you defined a text search

server named TM in the system administration client and a search index named TMINDEX associated with it, the value for the SearchIndex is TM-TMINDEX.

For an object that is to be indexed by DB2 Text Information Extender, 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 296).

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.

See LoadSampleTSQBICDL and LoadFolderTSQBICDL in the Samples directory for examples of how to load data, or create a folder and load data.

Important: When adding a part object to be indexed by a search engine, don’t set the RepType. Currently, the DB2 Text Information Extender works only with the default RepType FRN\$NULL.

Adding an XDO to be indexed by DB2 Text Information Extender: The following example shows how to add an XDO that is to be indexed:

Java

```
// ----- Declare variables for part ID, item ID, and file
int    partId = 20;
String itemId = "CPPIORH4JBIXWIY0";
String fileName = "g:\\test\\testsrch.txt";
try {
    DKDatastoreDL dsDL = new DKDatastoreDL(); // create datastore
    ...                                     // connect to datastore
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
    DKBlobDL axdo = new DKBlobDL(dsDL);      // create XDO
    DKPidXDODL apid = new DKPidXDODL();      // create PID
    apid.setPartId(partId);                  // set partId
    apid.setPrimaryId(itemId);               // set itemId
    axdo.setPidObject(apid);                // setPid to XDO
    axdo.setContentClass(DK_DL_CC_ASCII);    // set ContentClass to text

    // --- set the searchEngine
    DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo->setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
    ...
    // ----- Set file content to buffer area
    axdo.setContentFromClientFile(fileName);
    axdo.add();                             //add from buffer
    ...
    // ----- Display the partId after add
    System.out.println("after add partId = " + ((DKPidXDODL)
        (axdo.getPidObject())).getPartId());

    dsDL.disconnect();                      //disconnect from datastore
    dsDL.destroy();
}
// ----- Catch any exception
catch (...)
```

C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "N2JJBERBQFK@WTVL";
    repType = "FRN$NULL";
    partId = 10;
    if (argc == 1)
    {
        cout<<"invoke: indexPartxsDL <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: indexPartxsDL "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        cout<<"you enter: indexPartxsDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"you enter: indexPartxsDL "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }

    cout << "connecting Datastore" << endl;
    try
    {
        //replace following with your library server, user ID, password
        dsDL.connect("LIBSRVN","FRNADMIN","PASSWORD");

        cout << "datastore connected" << endl;

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

    // continued...
```


C++ (continued)

```
//--- set searchEngine -----
cout<<"set search engine and setToBeIndexed()"<<endl;
DKSearchEngineInfoDL aSrchEx;
aSrchEx.setSearchEngine("SM");
aSrchEx.setSearchIndex("TM-TMINDEX");
aSrchEx.setSearchInfo("ENU");
axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);
axdo->setToBeIndexed();
cout<<"setToBeIndexed() done..."<<endl;

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

Important: When adding a part object to be indexed by a search engine, don't set the RepType (representation type). The DB2 Text Information Extender works only with the default RepType FRN\$NULL.

Loading data to be indexed by DB2 Text Information Extender: To load data into DB2 Content Manager to be indexed by DB2 Text Information Extender, you must create both an index and a text search index.

Before you can create a text search index, the text search server must be running. Make sure that your environment is properly set up. To do so, you can customize and run the samples: TListCatalogDL and TListCatalogTS in the Samples directory.

To create parts in DB2 Content Manager that are indexed by the DB2 Text Information Extender, see "Working with extended data objects (XDOs)" on page 37.

After the data is loaded into DB2 Content Manager, use the wakeUpService function in the DKDatastoreDL class to place the documents on the document queue. This function takes a search engine name as a parameter.

Then from the DB2 Content Manager text search Administration window, complete the following steps:

1. Double-click the text search server.
2. Double-click the text search index.
3. Click **INDEX**.

This indexes the documents on the document query. After the indexing is complete, you can perform text search queries.

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

Using text structured document support

Text structured documents are composed of tagged text (for example, an HTML file). A document model defines the structure of the document, and the DB2 Text Information Extender can search on words or phrases between the tags.

You can perform text queries on structured documents as follows:

1. Create a document model. The document model contains sections, and each section includes the section name and document tag used. For example:

```
<HTML>
<HEAD>
<TITLE>Acme Corp<br></TITLE>
</HEAD>
<BODY>
<H1>Acme Corp<BR></H1>
<P><B>Acme Corp<BR></B><BR>
<P>John Smith <BR>
<P><ADDRESS>Acme Corporation<BR></ADDRESS>
<HR>
<H2>Acme Corp Business Objectives</H2>
<HR>
<P>
<H2><A NAME="Header_Test" HREF="#ToC_Test">Marketing</A></H2>
<P>We need to increase our time to market by 25%.
<P>We need to meet our customers needs.
</BODY>
</HTML>
```

2. Create a text search index that uses the DB2 Content Manager document model.
3. Set the indexing rules for the text search index and specify the default document format (for example, DK_TS_DOCFMT_HTML for HTML files)
4. Add parts objects to the DB2 Content Manager server.
5. Start the indexing process for the text search index.

This example shows how to list the document models defined in your system.

Java

```
// ----- Initialize the variables
DKSequentialCollection pCol = null;
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
dkIterator pIter = null;
DKDocModelTS pDocModel = null;
int ccsid = 0;
String strDocModelName = null;
int i = 0;

// ----- Create the datastore and connect
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect(srchSrv,"",' ');

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Get list of document models
pCol = (DKSequentialCollection) dsAdmin.listDocModels("");
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    pDocModel = (DKDocModelTS)pIter.next();
    strDocModelName = pDocModel.getName();
    ccsid = pDocModel.getCCSID();
}
dsTS.disconnect();
```

The complete sample application from which this example was taken (TListDocModelsTS.java) is available in the samples directory.

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKDocModelTS* docModel = 0;
DKSequentialCollection *pCol = 0;
long ccsid = 0;
DKString strDocModelName;
dkIterator *pIter = 0;
long i = 0;
DKAny a;

dsTS.connect(srchSrv,"","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// list document models
pCol = (DKSequentialCollection*)dsAdmin->listDocModels("");
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
    i++;
    docModel = (DKDocModelTS*)((void*)(*pIter->next()));
    strDocModelName = docModel->getName();
    ccsid = docModel->getCCSID();
    delete docModel;
}
delete pIter;
delete pCol;

dsTS.disconnect();
```

The complete sample application from which this example was taken (TListDocModelsTS.cpp) is available in the samples directory.

The following example shows how to create a document model:

Java

```
// ----- Create datastore and connect
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

// ----- Create an instance of a document model object
DKDocModelTS docModel = new DKDocModelTS();

// ----- Create 2 instances of a document section objects for the model
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- Describe the document model for text document structure
//          for files like tstruct.htm above
docModel.setCCSID(DK_TS_CCSID_00850);
docModel.setName(docModelName);
docSection.setName("SAMPITITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Create the document model
dsAdmin.createDocModel("", docModel);

dsTS.disconnect();
dsTS.destroy();

Refer to TCreateDocModelTS.java and TCreateStructDocIndexTS.java in the
samples directory for more examples.
```

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;

// create an instance of a document model object
DKDocModelTS* docModel = new DKDocModelTS();

// create 2 instances of a document section objects for the model
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// Describe the document model for text document structure
// for files like tstruct.htm above

docModel->setCCSID(DK_TS_CCSID_00850);
docModel->setName("SAMP CORPMOD");
docSection->setName("SAMP CORPTITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMP CORP BODY");
docSection2->setTag("BODY");
docModel->addDocSection(docSection2);

dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// create doc model
dsAdmin->createDocModel("", docModel);

// delete document model & sections
delete docModel;

dsTS.disconnect();

The complete sample application from which this example was taken
(TCreateDocModelTS.cpp) and (TCreateStructDocIndexTS.cpp) are available in
the samples directory.
```

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

Java

```
// ----- Create the datastore and index rules object
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
DKIndexingRulesTS indexRules = new DKIndexingRulesTS();

// ----- Create an instance of a document model object
DKDocModelTS docModel = new DKDocModelTS();

// ----- Create 2 instances of a document section objects for the model
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- Create the document model instance for indexing rules
DKDocModelTS docModel2 = new DKDocModelTS();
docModel2.setCCSID(DK_TS_CCSID_00850);
docModel2.setName("SAMPCORPMOD");

// ----- Describe the document model for text document structure
//           for files like tstruct.htm above
docModel.setCCSID(DK_TS_CCSID_00850);
docModel.setName("SAMPCORPMOD");
docSection.setName("SAMPITITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

// ----- Connect to the datastore
dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

DKSearchIndexDefTS pEnt = new DKSearchIndexDefTS((dkDatastore)dsTS);
pEnt.setName("TSTRUCT");
pEnt.setIndexType(DK_TS_INDEX_TYPE_PRECISE);
pEnt.setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);
pEnt.setLibraryId("LIBSUM");
pEnt.setLibraryClientServices("IMLLSCDL");
pEnt.setLibraryServerServices("IMLLSSDL");
String strIndexFileDir = "e:\\tsindex\\index\\tstruct";
// ----- For AIX us the following form for the file
//   String strIndexFileDir = "/home/cltadmin/tsindex/tstruct";
pEnt.setIndexDataArea(strIndexFileDir);
String strWorkFileDir = "e:\\tsindex\\work\\tstruct";
// ----- For AIX us the following form for the file
//   String strWorkFileDir = "/home/cltadmin/work/tstruct";
pEnt.setIndexWorkArea(strWorkFileDir);

// ----- Associate document model with index
pEnt.addDocModel(docModel);

// ----- Create text search index that supports sections
dsDef.add((dkEntityDef)pEnt);

// continued...
```

Java (continued)

```
indexRules.setIndexName("TSTRUCT");
indexRules.setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules.setDefaultDocModel(docModel2);

dsAdmin.setIndexingRules(indexRules);

dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (TCreateStructDocIndexTS.java) is available in the samples directory.

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexingRulesTS* indexRules = new DKIndexingRulesTS();

// create an instance of a document model object
DKDocModelTS* docModel = new DKDocModelTS();

// create 2 instances of a document section objects for the model
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// doc model instance for indexing rules
DKDocModelTS* docModel2 = new DKDocModelTS();
docModel2->setCCSID(DK_TS_CCSID_00850);
docModel2->setName("SAMPCORPMOD");

// Describe the document model for text document structure
// for files like tstruct.htm above

docModel->setCCSID(DK_TS_CCSID_00850);
docModel->setName("SAMPCORPMOD");
docSection->setName("SAMPCORPTITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMPCORPBODY");
docSection2->setTag("BODY");
docModel->addDocSection(docSection2);

// continued...
```


C++ (continued)

```
dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

DKSearchIndexDefTS* pEnt = new DKSearchIndexDefTS(&dsTS);

pEnt->setName("TSTRUCT");
pEnt->setIndexType(DK_TS_INDEX_TYPE_PRECISE);

// This index is text structure document section enabled
pEnt->setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);

pEnt->setLibraryId("LIBSUM");
pEnt->setLibraryClientServices("IMLLSCDL");
pEnt->setLibraryServerServices("IMLLSSDL");
DKString strIndexFileDir = "e:\\tsindex\\index\\tstruct";
//**** for AIX ****
//DKString strIndexFileDir = "/home/cltadmin/tsindex/index/tstruct";
pEnt->setIndexDataArea(strIndexFileDir);
DKString strWorkFileDir = "e:\\tsindex\\work\\tstruct";
//**** for AIX ****
//DKString strWorkFileDir = "/home/cltadmin/tsindex/work/tstruct";
pEnt->setIndexWorkArea(strWorkFileDir);

// Associate document model with index
pEnt->addDocModel(docModel);

// Create text search index that supports sections
dsDef->add(pEnt);

delete pEnt;

indexRules->setIndexName("TSTRUCT");
indexRules->setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules->setDefaultDocModel(docModel2);
dsAdmin->setIndexingRules(indexRules);

delete indexRules;

dsTS.disconnect();
```

The complete sample application from which this example was taken (TCreateStructDocIndexTS.cpp) is available in the samples directory.

The following example shows how to start the indexing process, which is asynchronous. Using the system administration program, you can start the indexing process and check its status.

Java

```
// ----- Declare datastore and administration
DKIndexFuncStatusTS pIndexFuncStatus = null;
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Start the indexing process
dsAdmin.startUpdateIndex(indexName);

// ----- Get indexing status
pIndexFuncStatus = dsAdmin.getIndexFunctionStatus(indexName,
                                                    DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS);
.... // Process the status as appropriate

// ----- Show the scheduled document queue
System.out.println("*getScheduledDocs "
    + pIndexFuncStatus.getScheduledDocs());

// ----- Show the primary document queue
System.out.println("*getDocsInPrimaryIndex "
    + pIndexFuncStatus.getDocsInPrimaryIndex());

// ----- Shows the secondary document queue
System.out.println("*getDocsInSecondaryIndex " +
    pIndexFuncStatus.getDocsInSecondaryIndex());
System.out.println("*getDocMessages "
    + pIndexFuncStatus.getDocMessages());
if (pIndexFuncStatus.isCompleted() == true)
{
    // ---- Processing if indexing is completed
}

if (pIndexFuncStatus.getReasonCode() != 0)
{
    dsAdmin.setIndexFunctionStatus(indexName,
        DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS, DK_TS_INDEX_ACTID_RESET);
}

dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (TIndexingTS.java) is available in the samples directory.

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexFuncStatusTS* pIndexFuncStatus = 0;

dsTS.connect(srchSrv,"","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// starts the indexing process
dsAdmin->startUpdateIndex(srchIndex);

// Get indexing status
pIndexFuncStatus = dsAdmin->getIndexFunctionStatus(srchIndex,
                                                    DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS);

cout << "***** index status *****" << endl;
cout << "*isCompleted " << pIndexFuncStatus->isCompleted() << endl;
cout << "*getEnabledId " << pIndexFuncStatus->getEnabledId() << endl;
cout << "*getReasonCode " << pIndexFuncStatus->getReasonCode()
    << endl;
cout << "*getFuncStopped " << pIndexFuncStatus->getFunctionStopped()
    << endl;
cout << "*getStartedLast " << pIndexFuncStatus->getStartedLast()
    << endl;
cout << "*getEndedLast " << pIndexFuncStatus->getEndedLast() << endl;
cout << "*getStartedExecution " << pIndexFuncStatus->getStartedExecution()
    << endl;
cout << "*getScheduledDocs " << pIndexFuncStatus->getScheduledDocs()
    << endl;
cout << "*getDocsInPrimaryIndex " << pIndexFuncStatus->getDocsInPrimaryIndex()
    << endl;
cout << "*getDocsInSecondIndex " << pIndexFuncStatus->getDocsInSecondIndex()
    << endl;
cout << "*getDocMessages " << pIndexFuncStatus->getDocMessages()
    << endl;
if (pIndexFuncStatus->isCompleted() == TRUE)
{
    // indexing completed
}
if (pIndexFuncStatus->getReasonCode() != 0)
{
    dsAdmin->setIndexFunctionStatus(srchIndex,
                                    DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS,DK_TS_INDEX_ACTID_RESET);
}
delete pIndexFuncStatus;
dsTS.disconnect();
```

The complete sample application from which this example was taken (TIndexingTS.cpp) is available in the samples directory.

SeeTCheckStatusTS in the Samples directory for an example of checking status. The example checks whether a queued request has been moved from the scheduled document queue to the primary or secondary queues. If an indexing error occurs, then you can check the imldiag.log file in the text search instance directory.

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

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNameValuePair parms[] = null;
// ----- Connect
dsTS.connect("TMMUF","","","");
// ----- Generate the query string
String cmd = "SEARCH=(COND=($CCSID=850," +
             "DOCMOD=(DOCMODNAME=SAMPCORPMOD," +
             "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));" +
             "OPTION=(SEARCH_INDEX=TMSTRUCT;MAX_RESULTS=5)";
// ----- Execute the query
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);

// ----- Process the results
.....
dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (TExecuteStructDocTS.java) is available in the samples directory.

C++

```
DKDatastoreTS dsTS;
dkResultSetCursor* pCur = 0;

dsTS.connect("TMMUF","","","");

DKString cmd = "SEARCH=(COND=($CCSID=850,";
cmd += "DOCMOD=(DOCMODNAME=SAMPCORPMOD,";
cmd += "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));";
cmd += "OPTION=(SEARCH_INDEX=TMSTRUCT;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 samples directory.

Searching images by content

The Image Search server can search for stored images when you specify the image type, or provide an example of the image.

Figure 16 on page 296 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.

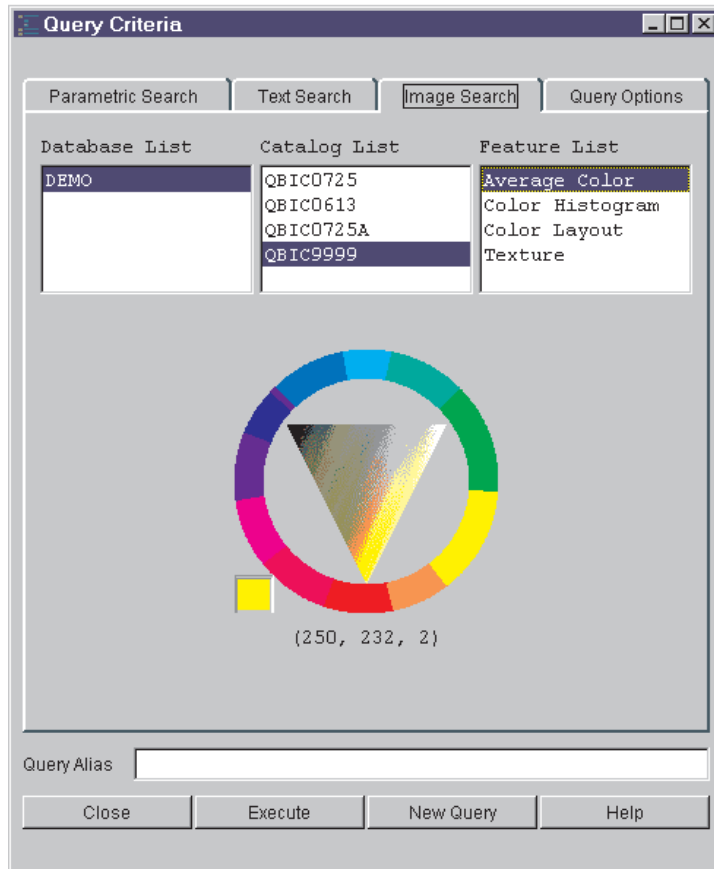


Figure 16. Image search sample client

Understanding image search terms and concepts

This section describes the image search components: the server, databases, catalogs, and the relationship of the image search server to earlier DB2 Content Manager. It also describes *features* that are the searchable visual characteristics of images.

Understanding image search servers, databases, and catalogs: Earlier DB2 Content Manager uses an image search server to search for images. DB2 Content Manager applications store image objects in the object server. The image search server analyzes and indexes the image information. The image search server does not store images themselves.

A content server defined by a DKDatastoreQBIC object represents the image search server. The results of an image search include identifiers (item IDs) that describe the location of the image in the DB2 Content Manager server. You can use these identifiers with other results, such as the part number and RepType, to retrieve the image.

You can perform queries on the content server. However, the content server for image search does not support add, update, retrieve, and delete operations. Figure 17 on page 297 shows an example of an image search server.

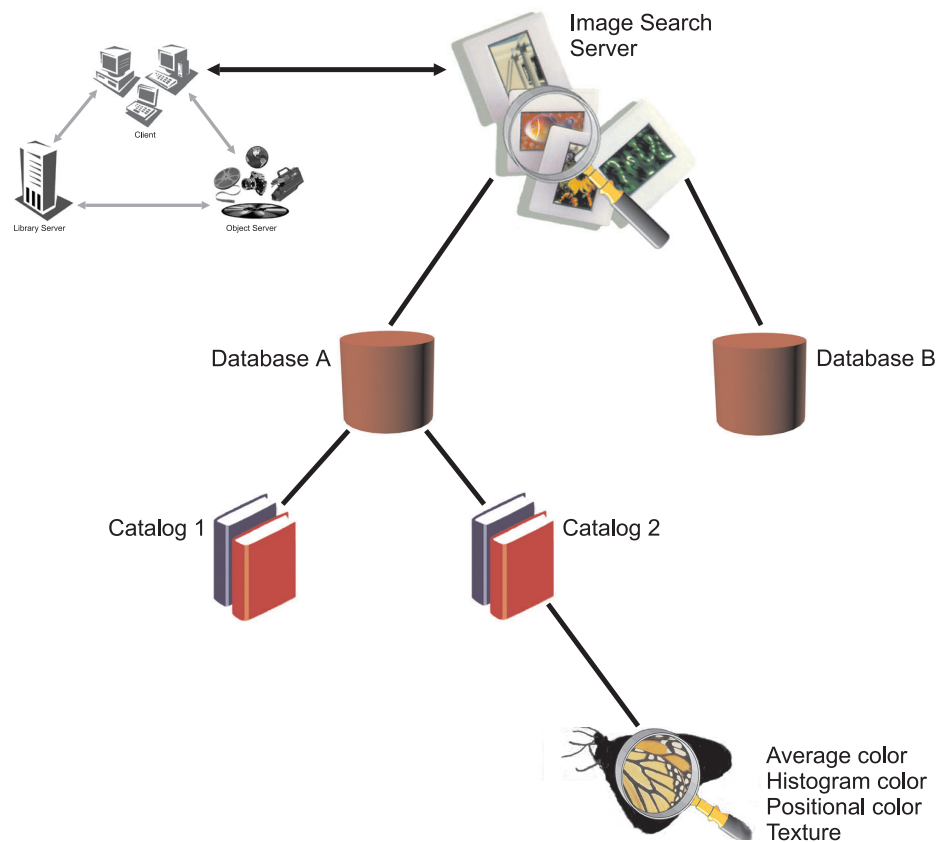


Figure 17. An image search server in an earlier DB2 Content Manager system

The image search server can contain one or more databases. Each database can contain one or more catalogs, which hold information about the visual features of images. These four image search features are:

- Average color.
- Histogram color.
- Positional color (color layout).
- Texture.

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

Average color Use average color to search for images that have a predominant color. Images with similar predominant colors have similar average colors. For example, images that contain equal portions of red and yellow have an average color of orange.

QbColorFeatureClass 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 (color layout)

Positional colors measure the average color value for the pixels in a specified area of an image. For example, images with bright red objects in the middle have a positional color of bright red.

QbDrawFeatureClass is the feature name for positional color.

Texture

Use texture to search for images that have a particular pattern. Texture measures the coarseness, contrast, and directionality of an image. Coarseness indicates the size of repeating items in an image. Contrast identifies the brightness variations in an image. Directionality indicates whether a direction predominates in an image. For example, an image of a wood grain has a similar texture to other images that contain a wood grain.

QbTextureFeatureClass is the feature name for texture.

Using image search applications

Image search client applications create image queries, run them, and then evaluate the information returned by the image search server. Before an application can search images by content, the images must be indexed, and the content information must be stored in an image search database.

Restriction: You cannot index existing images in your object server. You can index only the images you create in your object server after you install the image search server and client. Figure 18 shows an example of the client and retrieve images.

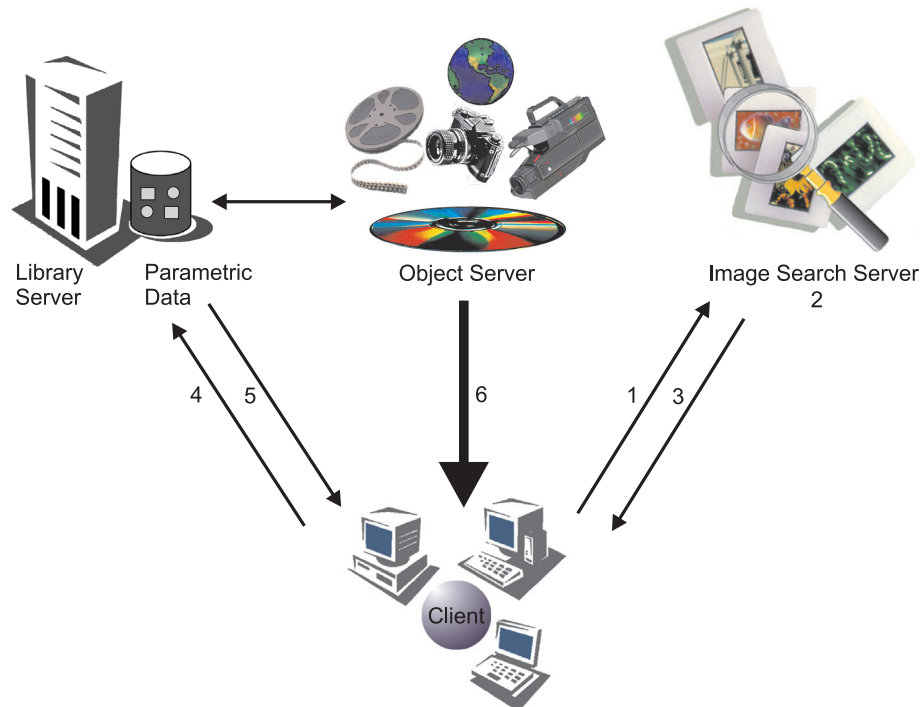


Figure 18. How image search clients search and retrieve images

To perform an image search:

1. A client builds a QBIC query string and sends it to an image search server.

2. Image search server receives the query string and searches the cataloged images for matches.
3. Client receives the matches as a list of identifiers. The identifier for each matching image consists of the:
 - Item ID.
 - Part number.
 - RepType.
 - Rank.
4. Client requests the image part and index information from a library server.
5. Library server returns index information, such as a text description, to the client. The library server also requests that an object server send specified image parts to the client.
6. Object server sends image parts and the client acknowledges receiving them.

Creating queries

When you create queries, you construct a query string that the application passes to the image search server. An image query is a character string that specifies the search criteria. The search criteria consists of:

An image query is a character string that specifies the search criteria. The search criteria consists of:

Feature name The features used in the search.

Feature value The values of those features. Table 27 on page 300 shows the image search feature names and the values that can be passed in a query string.

Feature weight

The relative weight or emphasis placed on each feature. The weight of a feature indicates the emphasis that the image search server places on the feature when calculating similarity scores and returning results for a query. The higher the specified weight, the greater the emphasis.

Maximum results

In addition to defining the type of images a query will look for, you can specify the maximum number of matches that the query will return.

A query string has the form: *feature_name value*, where *feature_name* is an image search feature name, and *value* is a value associated with the feature. If you use more than one feature in a query, then you must specify a feature name-value pair for each feature. The string "and" separates each pair.

Image search queries have the following syntax:

```
feature_name value
feature_name value weight
```

You cannot repeat a feature within a single query. You can specify multiple features in a query. When you specify multiple features in a query, you can assign a weight to one or more of the features. Queries that include the emphasis for each feature have the form: *feature_name value weight*, where *feature_name* is an image search feature name, *value* is a value associated with the feature, and *weight* is the weight assigned to the feature. *weight* is the combination of the keyword weight, an equal sign (=), and a real number greater than 0.0.

You can also specify the maximum number of matches that a query returns. To specify the maximum results, append `max_results` to your query. `max_results` consists of the keyword `max`, an equal sign (`=`), and an integer greater than 0. Table 27 describes feature names and values.

Table 27. Image search query: valid feature values

Feature name	Values
QbColorFeatureClass or QbColor	<p>color = < rgbValue , rgbValue , rgbValue > where rgbValue is an integer from 0 to 255.</p> <p>file = < fileLocation , " fileName " > where fileLocation is either <code>server</code> or <code>client</code>, 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:</p> <p>QbColorFeatureClass color= <50, 50, 50> and QbTextureFeatureClass file=<client, "\patterns\pattern1.gif"> weight=2.0</p>
QbColorHistogramFeatureClass or QbHistogram	<p>histogram = < histList > where histList consists of one or more histClause separated by a comma (,).</p> <p>A histClause is specified as (histValue, rgbValue , rgbValue , rgbValue), where histValue is an integer from 1 to 100 (a percentage value), and rgbValue is an integer from 0 to 255.</p> <p>file = < fileLocation , " fileName " > where fileLocation is either <code>server</code> or <code>client</code>, fileName is the complete file path specified in the format appropriate for the system on which the file resides.</p>

Table 27. Image search query: valid feature values (continued)

Feature name	Values
QbDrawFeatureClass or QbDraw	<p>description = < " descString " > where descString is a special encoded string describing a picker file. Format of the description string:</p> <ol style="list-style-type: none"> 1. Dw,h specifies the outer dimensions of the image itself with width w and height h. 2. Rx,y,w,h,r,g,b specifies that a rectangle of width w and height h is to be positioned with its upper left corner at the coordinates (x,y)—with respect to an origin in the upper left corner of the image rectangle—and this rectangle should have color values r (red), g (green), and b (blue). 3. Use the colon character (:) is used as a separator. <p>For example, you can search for color layout (QbDrawFeatureClass) described by the description string: QbDrawFeatureClass description= <"D100,50:R0,0,50,50,255,0,0"</p> <p>file = < fileLocation , " fileName " > where fileLocation is either server or client, fileName is the complete file path specified in the format appropriate for the system on which the file resides.</p>
QbTextureFeatureClass or QbTexture	<p>file = < fileLocation , " fileName " > where fileLocation is either server or client, fileName is the complete file path specified in the format appropriate for the system on which the file resides.</p>

Query examples:

1. Search for average color red:
QbColorFeatureClass color=<255,0,0>
2. Search using a histogram of three colors, 10% red, 50% blue, and 40% green:
QbColorHistogramFeatureClass histogram=
<(10, 255, 0, 0) (50, 0, 255, 0), (40, 0, 0, 255)>
3. Search using an average color and a texture value. The texture value is provided by an image in a file on the client. The weight of the texture is twice that of the average color:
QbColorFeatureClass color=
<50, 50, 50> and QbTextureFeatureClass file=<client, "\patterns\pattern1.gif">
weight=2.0
4. Search for the described color layout:
QbDrawFeatureClass description=<"D100,50:R0,0,50,50,255,0,0">
5. Search for average color red and limiting the returned matches to five:
QbColorFeatureClass color=<255,0,0> and max=5

Running queries and evaluating search results

Applications use the image search API to issue queries and evaluate search results. If information in the image search database matches the image search criteria, then an identifier of the matching image or images is returned. This identifier is a dynamic data object (DDO) that corresponds to an image part inside a DB2 Content Manager object.

Establishing a connection in QBIC

Image search provides functions for connecting and disconnecting to the content server. The following example shows how to connect to an image search server named QBICSRV using the user ID QBICUSER and the password PASSWORD.

Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...           // Process as appropriate
dsQBIC.disconnect();
```

The complete sample application from which this example was taken (TConnectQBIC.java) is available in the samples directory.

C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...           // do some work
dsQBIC.disconnect();
```

The complete sample application from which this example was taken (TConnectQBIC.cpp) is available in the samples directory.

The image search connection allows an application to connect to an image search server.

After connecting, your program can use functions that access the image search server, except for the functions that are not related to image search catalogs, such as `listDatabases`. An `openCatalog` function is required to open a catalog for processing. A `closeCatalog` function is called after processing is done. The following example shows how to connect, open a catalog, close the catalog, and disconnect.

Java

```
// ----- Create a QBIC datastore and connect
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
// ----- open the catalog
dsQBIC.openCatalog("DEMO", "QBIC0725");
...           // Do some processing
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "QBIC0725");
...                               // do some work
dsQBIC.closeCatalog();
dsQBIC.disconnect();
```

Listing image search servers

The image search server provides a function for listing the image search servers that it can connect to. The following example shows how to retrieve (in a `DKSequentialCollection` object) the list of servers that contain `DKServerInfoQBIC` objects. After you get a `DKServerInfoQBIC` object, you can retrieve the server name, the host name, and the port number.

Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
.....
DKServerInfoQBIC pSV = null;
String strServerName = null;
String strHostName = null;
String strPortNumber = null;
pCol = (DKSequentialCollection)dsQBIC.listDataSources();
iter = pCol.createIterator();
while (iter.more()) {
    srvDef = (DKServerDefQBIC)iter.next();
    .....    // Process each server as appropriate
}
```

The complete sample application from which this example was taken (`TListCatalogQBIC.java`) is available in the samples directory.

C++

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKServerDefQBIC *pSV = 0;
DKString strServerName;
DKAny a;
long i = 0;
cout << "list servers" << endl;
a = dsQBIC.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
    i++;
    pSV = (DKServerDefQBIC*)((void*)(*pIter->next()));
    strServerName = pSV->getName();
    cout << "Server Name [" << i << "] - " << strServerName << endl;
    delete pSV;
}
delete pIter;
delete pCol;
```

The complete sample application from which this example was taken (TListCatalogQBIC.cpp) is available in the samples directory.

Listing image search databases, catalogs, and features

DKDatastoreQBIC provides a function for listing all of the image search databases, catalogs, and features on an image search server. The list is returned in a DKSequentialCollection object that contains DKIndexQBIC objects. After you get a DKIndexQBIC object, you can retrieve the database, catalog, and feature name. The following example shows how to retrieve the list of databases, catalogs, and features.

Java

```
// ----- Create the datastore and connect
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");

// ---- Get the list of servers
col = (DKSequentialCollection)dsQBIC.listDataSources();
iter = col.createIterator();
while (iter.more()) {
    srvDef = (DKServerDefQBIC)iter.next();
    .....    // Process each server as appropriate
}

// ----- Get the list of QBIC Databases
col = (DKSequentialCollection)dsQBIC.listEntities();
iter = col.createIterator();
while (iter.more()){
    dbDef = (DKDatabaseDefQBIC)iter.next();
    // ----- Get the list of catalogs for the database
    col2 = (DKSequentialCollection)dbDef.listSubEntities();
    iter2 = col2.createIterator();
    while (iter2.more()){
        catDef = (DKCatalogDefQBIC)iter2.next();
        // ----- Get the list of features for the catalog
        col3 = (DKSequentialCollection)catDef.listAttrs();
        iter3 = col3.createIterator();
        while (iter3.more()){
            featDef = (DKFeatureDefQBIC)iter3.next();
            .... // Process the features as appropriate
        }
    }
}
dsQBIC.disconnect();
dsQBIC.destroy();
.....
```

The complete sample application from which this example was taken (TListCatalogQBIC.java) is available in the samples directory.

C++

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKSequentialCollection *pCol2 = 0;
dkIterator *pIter2 = 0;
DKSequentialCollection *pCol3 = 0;
dkIterator *pIter3 = 0;
DKDatabaseDefQBIC *pEntDB = 0;
DKCatalogDefQBIC *pEntCat = 0;
DKString strCatName;
DKString strDBName;
DKString strFeatName;
DKFeatureDefQBIC *pAttr = 0;
DKAny a;
DKAny *pA = 0;
long i = 0;
long j = 0;
long k = 0;
cout << "connecting to datastore" << endl;
dsQBIC.connect("QBICSRV","USERID","PW");
cout << "list databases " << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsQBIC.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
    i++;
    pEntDB = (DKDatabaseDefQBIC*)((void*)(*pIter->next()));
    strDBName = pEntDB->getName();
    cout << "database name [" << i << "] - " << strDBName << endl;
    cout << " list catalogs for DB " << strDBName << endl;
    pCol2=(DKSequentialCollection*)((dkCollection*)pEntDB->listSubEntities());
    pIter2 = pCol2->createIterator();
    j = 0;
    while (pIter2->more() == TRUE)
    {
        j++;
        pA = pIter2->next();
        pEntCat = (DKCatalogDefQBIC*) pA->value();
        strCatName = pEntCat->getName();
        cout << "catalog name [" << j << "] - " << strCatName << endl;
        pCol3=(DKSequentialCollection*)((dkCollection*)pEntCat->listAttrs());
        pIter3 = pCol3->createIterator();
        k = 0;
        while (pIter3->more() == TRUE)
        {
            k++;
            pA = pIter3->next();
            pAttr = (DKFeatureDefQBIC*) pA->value();
            cout << "    Attribute name [" << k << "] - "
                << pAttr->getName() << endl;
            cout << "        datastoreName " << pAttr->datastoreName()
                << endl;
            cout << "        datastoreType " << pAttr->datastoreType()
                << endl;
            cout << "        attributeOf " << pAttr->getEntityName()
                << endl;
            delete pAttr;
        }
    }
}
// continued...
```

C++ (continued)

```
        delete pIter3;
        delete pCol3;
        delete pEntCat;
    }
    cout << " " << j << " features listed for catalog: "
        << strCatName << endl;
    delete pIter2;
    delete pCol2;
    delete pEntDB;
}
delete pIter;
delete pCol;
cout << i << " databases listed" << endl;
dsQBIC.disconnect();
```

The complete sample application from which this example was taken (TListCatalogQBIC.cpp) is available in the samples directory.

Representing image search information with a DDO

A DDO associated with DKDatastoreQBIC contains specific information for representing image search results. A DDO resulting from an image query corresponds to an image part inside an item; it has the following set of standard attributes:

DKDLITEMID

The item ID for the item to which this image part belongs. Use the item ID to retrieve the whole item from the content server.

DKPARTNO

An integer part number of this image part. Use this with the item ID to retrieve this part from the content server.

DKREPTYPE

A string for representation type (RepType). The default value is FRN\$NULL. This attribute is reserved.

DKRANK

An integer rank indicating the relevance of this part to the results set of the image query. The rank is within the range 0 to 100. A higher rank means a better match.

The PID for an image search DDO has the following information:

content server type

QBIC.

content server name

The server name used to connect to the content server.

ID

The zero-based sequence number of the DDO in the results set.

As a convention, the attribute value is always an object.

Working with image queries

This section describes how to run and evaluate image queries.

Running an image query

Using an instance of `dkQuery` from `DKDatastoreQBIC`, you can create a query object to run the query and obtain the results. The following example shows how to create an image query object and run it. After you run a query, the results are returned in a `DKResults` collection.

Java

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

...    // Process as appropriate

// ----- Create the query and run it
dkQuery pQry = dsQBIC.createQuery(cmd, DK_IMAGE_QL_TYPE, parms);
pQry.execute(parms);
// ----- Get the results and process
DKResults pResults = (DKResults)pQry.result();
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
    item = (DKDDO)pIter.next();
    // Process the DKDDO
}
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
...
```

The complete sample application from which this example was taken (`SampleIQryQBIC.java`) is available in the samples directory.

C++

```
DKDatastoreQBIC* dsQBIC;
dsQBIC = new DKDatastoreQBIC();
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");
dsQBIC->openCatalog("DEMO", "qbic0725");
DKAny* element;
DKDDO* item;
DKString cmd = "QbColor color=<255, 0, 0>";
dkQuery* pQry = dsQBIC->createQuery(cmd);
pQry->execute();
DKAny any = pQry->result();
DKResults* pResults = (DKResults*)((dkCollection*)any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more())
{
    element = pIter->next();
    item = (DKDDO*)element->value();
    // Process the DKDDO
    ...
}
delete pIter;
delete pResults;
delete pQry;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

The complete sample application from which this example was taken (TSampleIQryQBIC.cpp) is available in the samples directory.

Running an image query from the content server

As an alternative, you can use the execute function of DKDatastoreQBIC to run a query. The results are returned in a dkResultSetCursor object. The following example shows how to run an image query on the content server. Results are returned in a dkResultSetCursor object.

Java

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

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");
// ----- Execute the query from the datastore
dkResultSetCursor pCur = dsQBIC.execute(cmd, DK_IMAGE_QL_TYPE, parms);
while (pCur.isValid())
{
    item = pCur.fetchNext();
    ....    // Process the DKDDO
}
pCur.destroy();
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

The complete sample application from which this example was taken (TExecuteQBIC.java) is available in the samples directory.

C++

```
DKDatastoreQBIC* dsQBIC;
dsQBIC = new DKDatastoreQBIC();
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");
cout << "datastore connected" << endl;
dsQBIC->openCatalog("DEMO", "qbic0725");
DKString cmd = "QbColorFeatureClass color=<255, 0, 0>";
dkResultSetCursor* pCur = dsQBIC->execute(cmd);
DKDDO* item = 0;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        // Process the DKDDO
        ...
        delete item;
    }
}
delete pCur;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

The complete sample application from which this example was taken (TExecuteQBIC.cpp) is available in the samples directory.

Evaluating an image query from the content server

DKDatastoreQBIC also provides a function to evaluate a query. The following example shows how to evaluate an image query from the content server. Results are returned in a DKResults collection.

Java

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

// ----- Use evaluate to run the query
DKResults pResults=(DKResults) dsQBIC.evaluate(cmd,DK_IMAGE_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
    item = (DKDDO)pIter.next();
    ... // Process the DKDDO
}
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

C++

```
DKDatastoreQBIC* dsQBIC;  
dsQBIC = new DKDatastoreQBIC();  
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");  
dsQBIC->openCatalog("DEMO", "qbic0725");  
DKAny* element;  
DKDDO* item;  
DKString cmd = "QbColor color=<255, 0, 0>";  
DKAny any = dsQBIC->evaluate(cmd);  
DKResults* pResults = (DKResults*)((dkCollection*)any);  
dkIterator* pIter = pResults->createIterator();  
while (pIter->more())  
{  
    element = pIter->next();  
    item = (DKDDO*)element->value();  
    // Process the DKDDO  
    ...  
}  
delete pIter;  
delete pResults;  
dsQBIC->closeCatalog();  
dsQBIC->disconnect();
```

Using the image search engine

You can use the image search server to specify a query based on one of the following features: average color, color percentages, color layout, and textures. You can also specify multiple features in a query. The query results contain the item ID, part number, representation type, and ranking information. You can use this information to create an XDO for retrieving the image contents.

Loading data to be indexed for image search

To load data into a DB2 Content Manager server to be indexed by the image search server, you must create a DB2 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 DB2 Content Manager, you can place the image in the image queue. In the system administration program, select **Process Image Queue**. After the indexing is complete, you can run image searches.

Indexing an existing XDO using search engines

You can index an existing XDO using a specified search engine. The following example calls the `setToBeIndexed` function of the `DKBlobDL` class.

Java

```
try
{
    // ----- Create the datastore and connect
    DKDatastoreDL dsDL = new DKDatastoreDL();
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");

    // ----- Create the XDO and PID and set attributes
    DKBlobDL axdo = new DKBlobDL(dsDL);
    DKPidXDODL apid = new DKPidXDODL();
    apid.setPartId(partId);
    apid.setPrimaryId(itemId);
    axdo.setPidObject(apid);

    // ----- Set search engine information
    DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo.setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
    // ----- Call setToBeIndexed on the XDO
    axdo.setToBeIndexed();

    dsDL.disconnect();
    dsDL.destroy();
}
catch (DKException exc)
{
    ... // Handle the DKException
}
catch (Exception exc)
{
    ... // Handle the Exception
}
```

C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "N2JJBERBQFK@WTVL";
    repType = "FRN$NULL";
    partId = 10;
    if (argc == 1)
    {
        cout<<"invoke: indexPartxs <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: indexPartxs "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }

    cout << "connecting Datastore" << endl;
    try
    {
        //replace following with your library server, userid, password
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setId(itemId);
        axdo ->setPid(apid);
        axdo ->setRepType(repType);
        cout<<"itemId= "<<(axdo->getPid())->getId()<<endl;
        cout<<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;
        cout<<"repType= "<<axdo->getRepType()<<endl;
    }

    // continued...
```

C++ (continued)

```
//--- set searchEngine -----
cout<<"set search engine and setToBeIndexed()"<<endl;
DKSearchEngineInfoDL aSrchEx;
aSrchEx.setSearchEngine("SM");
aSrchEx.setSearchIndex("TM-TMINDEX");
aSrchEx.setSearchInfo("ENU");
axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);
axdo->setToBeIndexed();
cout<<"setToBeIndexed() done..."<<endl;

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

Using combined query

Use a *combined query* to execute a combination of parametric and text queries, with or without a scope. A *scope* is a DKResults object formed from a previous parametric or text query. The result is an intersection between the scopes and the results of each query. Therefore, if you are not careful when formulating the query and including scopes, individual query results might not intersect and the result of the combined query is empty.

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

Restriction: For each query in a combined query, you must use a different connection to the search engine; you cannot route multiple queries through the same connection.

Combined parametric and text queries

To run a combined query made up of one parametric and one text query, without a scope, you must create a combined query object and pass the two queries as input parameters to be run by the combined query. For example:

Java

```
// ----- Create a pre-Version 8.1 Content Manager datastore and connect
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Create a text search datastore and connect
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' '); // TM is a local alias for
...                          // the DB2 Text Information Extender server

// ----- Generate the parametric query string and create the query
String pquery = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(DLSEARCH_Date > 1994));";
DKParametricQuery pq =
    (DKParametricQuery) dsDL.createQuery(pquery, DK_CM_PARAMETRIC_QL_TYPE, null);

// ----- Generate the text query string and create the query
String tquery = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery, DK_CM_TEXT_QL_TYPE, null);

// ----- Create the combined query
DKCombinedQuery cq = new DKCombinedQuery();

// ----- Package the queries in DKNVPair as input parameters
DKNVPair par[] = new DKNVPair[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].setName(DK_PARM_END); // to signal the end of parameter list

// ----- Execute the combined query
cq.execute(par);

// ----- Get the results
DKResults res = (DKResults) cq.result();
if (res != null) {
    ... // process the results
}
```


C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

DKDatastoreTS dsTS;
// TM is a local alias for the DB2 Text Information Extender server
dsTS.connect("TM",""," ' ');
// create a parametric query
DKString pquery="SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994));";
DKParametricQuery* pq =
    (DKParametricQuery*) dsDL.createQuery(pquery,DK_PARAMETRIC_QL_TYPE, NULL);

// create a text query
DKString tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery* tq =
    (DKTextQuery*) dsTS.createQuery(tquery,DK_TEXT_QL_TYPE, NULL);

// create a combined query
DKCombinedQuery* cq = new DKCombinedQuery();

// package the queries in DKNVPair as input parameters
DKNVPair par[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
// to signal the end of parameter list
par[2].setName(DK_PARM_END);

// execute the combined query
cq->execute(par);

// get the results
DKAny any = cq->result();
DKResults* res = (DKResults*) any.value();
if (res != NULL) {
    // process the results
    ...
}
```

The last if statement is necessary to ensure that the DKResults object is not null.

Using a scope

If you have a DKResults object that you want to use as the scope, pass it as an additional query parameter. The following example illustrates using a DKResults object to function as a scope for a combined query:

Java

```
// ----- This scope is the result of a parametric query
DKResults scope;
// ----- This scope is the result of a previous text query
DKResults tscope;

// ----- Package the query in array of DKNVPairs as input parameters
DKNVPair par[] = new DKNVPair[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);

// ----- Execute the combined query
cq.execute(par);
....
```

C++

```
DKResults* scope;    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some parametric query
DKResults* tscope    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some text query

...
// package the query in DKNVPair as input parameters
DKNVPair par[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);
// execute the combined query
cq->execute(par);
...
```

The results of one combined query can also be used as a scope for another combined query, and sometimes you can query the results.

Ranking

If the combined query contains at least one text query, then the resulting DDO has the attribute DKRANK. This attribute is not stored, but is computed each time by the DB2 Text Information Extender. The value of the rank corresponds to the highest rank of the part in the document that satisfies the text query conditions.

Tips

If you have several parametric queries and scopes, it is more efficient to run one complete query. This is also true for text queries.

The query option "MAX_RESULTS=nn" limits the number of returned results. Usually, this option is more applicable to text queries, because the result is sorted in descending order by rank. If this option is set to 10, for example, it means that the caller only wants the 10 highest matching results.

The meaning of the query option "MAX_RESULTS=nn" is different for parametric queries. Because there is no notion of rank, the caller gets the first 10 results. The results are intersected with the result from the text query. Therefore, when combining parametric and text queries, it is advisable not to specify the query option "MAX_RESULTS=nn" for the parametric query.

Understanding the earlier DB2 Content Manager workflow and workbasket functions

This section describes the earlier DB2 Content Manager workflow and workbasket functions.

Understanding the earlier DB2 Content Manager workflow service

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

The workflow model in DB2 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 DB2 Information Integrator for Content APIs provide classes to work with DB2 Content Manager workflow.

The DKWorkflowServiceDL class represents the workflow service of DB2 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 DB2 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 DB2 Content Manager server named LIBSRVRN, using the user ID FRNADMIN and the password PASSWORD.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
...           // Process as appropriate
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TListWorkflowWFS.java) is available in the samples directory.

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
...           // do some work
dsDL.disconnect();
```

The complete sample application from which this example was taken (TListWorkflowWFS.cpp) is available in the samples directory.

Creating a workflow

Use DKWorkflowServiceDL to create a workflow. To do this, you typically complete the following six steps:

1. Create an instance of DKWorkflowDL.
2. Set the workflow name ("GOLF").
3. Set the workbasket sequence ("NULL") to indicate that this workflow contains no workbaskets.
4. Set the privilege ("All Privileges").
5. Set the disposition (DK_WF_SAVE_HISTORY).
6. Call the add function add ().

The example follows the six steps to create a workflow.

Java

```
// ----- Create the datastore and the CM workflow services
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);

// ----- Set the access option and connect
Object input_option = new Integer(DK_SS_CONFIG);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");

// ----- Create the CM workflow
DKWorkflowDL newwf = new DKWorkflowDL(wfDL);
newwf.setName("Process claim");
newwf.setWorkBasketSequence((dkCollection *)NULL);
newwf.setAccessList("All Privileges");
newwf.setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf.add();
... // Processing as appropriate
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TCreateDelWorkflow.java) is available in the samples directory.

C++

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_DL_OPT_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkflowDL * newwf = new DKWorkflowDL(&wfDL);
newwf->setName("GOLF");
newwf->setWorkBasketSequence((dkCollection *)NULL);
newwf->setAccessList("All Privileges");
newwf->setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf->add();
... // do some work
dsDL.disconnect();
```

The complete sample application from which this example was taken (TCreateDelWorkflowWFS.cpp) is available in the samples directory

Important: If you connect to the content server as a normal user (DK_SS_NORMAL), you do not get the workflow defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

Listing workflows

DKWorkflowServiceDL provides a function for listing the workflows in the system as shown in the following example. The list is returned in a sequential collection of DKWorkflowDL objects.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Get a list of the CM workflows
DKSequentialCollection wfList=(DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
{
    dkIterator pIter = wfList.createIterator();
    DKWorkflowDL pwf1;
    while (pIter.more())
    {
        pwf1 = (DKWorkflowDL)pIter.next();
        pwf1->retrieve();
        ...                // Process as appropriate
    }
}
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TListWorkFlowWFS.java) is available in the samples directory.

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 =
    (DKSequentialCollection *)wfDL.listWorkFlows();
if (wfList1 != NULL)
{
    dkIterator * pIter1 = wfList1->createIterator();
    DKWorkflowDL * pwf1;
    while (pIter1->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 samples directory.

Creating a DB2 Content Manager workbasket

Use DKWorkflowServiceDL to create a workbasket. To do this, you typically complete the following steps:

1. Create an instance of DKWorkBasketDL.
2. Set the workbasket name (Hot Items).
3. Set the privilege (All Privileges).
4. Call the add function.

The following example follows these steps to create a workbasket. If you connect to the content server as a normal user (DK_SS_NORMAL), you do not get the workbasket defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
Object input_option = new Integer(DK_SS_CONFIG);
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Create the CM workbasket and set properties
DKWorkBasketDL newwb = new DKWorkBasketDL(wfDL);
newwb.setName("Hot Items");
newwb.setAccessList("All Privileges");
newwb.add();
...      // Process as appropriate
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TCreateDelWorkBasket.java) is available in the samples directory.

C++

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_DL_OPT_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkBasketDL * newwb = new DKWorkBasketDL(&wfDL);
newwb->setName("Hot Items");
newwb->setAccessList("All Privileges");
newwb->add();
...                                     // do some work
dsDL.disconnect();
```

The complete sample application from which this example was taken (TCreateDelWorkBasket.cpp) is available in the samples directory.

Listing workbaskets

DKWorkflowServiceDL provides a function for listing the workbaskets in the system as shown in the following example. The list is returned in a sequential collection of DKWorkBasketDL objects.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection wbList=(DKSequentialCollection)wfDL.listWorkBaskets();
if (wbList != null)
{
    dkIterator pIter = wbList.createIterator();
    DKWorkBasketDL pwbl;
    while (pIter.hasMore())
    {
        pwbl = (DKWorkBasketDL)pIter.next();
        pwbl->retrieve();
        ... // do some work
    }
}
dsDL.disconnect();
dsDL.destroy();
```

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

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 =
(DKSequentialCollection *)wfDL.listWorkBaskets();
if (wfList1 != NULL)
{
    dkIterator * pIter1 = wfList1->createIterator();
    DKWorkBasketDL * pwbl;
    while (pIter1->more())
    {
        pwbl = (DKWorkBasketDL *)((void*)(*pIter1->next()));
        pwbl->retrieve();
        ... // do some work
        delete pwbl;
    }
}
dsDL.disconnect();
```

The complete sample application from which this example was taken (TListWorkBasketWFS.cpp) is available in the samples directory.

Listing items in an earlier DB2 Content Manager workflow

DKWorkflowServiceDL provides a function for listing the item IDs of the items in a workflow as shown in the following example. The list is returned in a sequential collection of DKString objects.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Get the list of CM workflows
DKSequentialCollection 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 samples directory.

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkflowDL * wf = new DKWorkflowDL(&wfDL, (char *)itemIDWF);
wf->retrieve;
DKSequentialCollection * pColDoc1 =
    (DKSequentialCollection *)wf->listItemIDs();
if (pColDoc1 != NULL)
{
    dkIterator* pIterDoc1 = pColDoc1->createIterator();
    DKString DocID1;
    while (pIterDoc1->more() == TRUE)
    {
        DocID1 = (DKString)(*pIterDoc1->next());
        ... // do some work
    }
}
dsDL.disconnect();
```

The complete sample application from which this example was taken (TListItemsWFS.cpp) is available in the samples directory.

Executing an earlier DB2 Content Manager workflow

DKWorkflowServiceDL provides functions for executing a workflow. The following example demonstrates how to start an item in a workflow, how to route an item to a workbasket, and how to complete an item in a workflow. To use this sample you must modify it to:

- Use a valid item ID instead of EP8L80R9MHH##QES.
- Use a valid workflow ID instead of HI7MOPALUPFQ1U47.
- Use a valid workbasket ID instead of E3PP1UZ0ZUFQ1U3M.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
DKString itemID = new String("EP8L80R9MHH##QES");
DKString itemIDWF = new String("HI7MOPALUPFQ1U47");
DKString itemIDWB = new String("E3PP1UZ0ZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkflowItem(itemID,           // itemID
                      itemIDWF,         // itemIDWB
                      NULL,              // default (1st workbasket)
                      TRUE,              // overload
                      DK_WIP_DEFAULT_PRIORITY // initial_priority
                      );
...
wfDL.routeWipItem(itemID,               // itemID
                  itemIDWF,             // itemIDWB
                  TRUE,                 // overload
                  DK_NO_PRIORITY_CHANGE // initial_priority
                  );
...
wfDL.completeWorkflowItem(itemID);
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TProcessWFS.java) is available in the samples directory.

C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemID = DKString("EP8L80R9MHH#QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZ0ZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkflowItem(itemID,           // itemID
                      itemIDWF,         // itemIDWB
                      NULL,              // default(1st workbasket)
                      TRUE,              // overload
                      DK_WIP_DEFAULT_PRIORITY // initial_priority
                      );
...                                     // do some work
wfDL.routeWipItem(itemID,               // itemID
                  itemIDWF,             // itemIDWB
                  TRUE,                 // overload
                  DK_NO_PRIORITY_CHANGE // initial_priority
                  );
...                                     // do some work
wfDL.completeWorkflowItem(itemID);
dsDL.disconnect();
```

The complete sample application from which this example was taken (TProcessWFS.cpp) is available in the samples directory.

Working with OnDemand

Information Integrator for Content provides a connector and related classes for accessing content on Content Manager OnDemand servers. The OnDemand classes and APIs allow you to:

- Connect to and disconnect from OnDemand servers.
- List application groups and application group fields.
- List OnDemand folders.
- Query an application group.
- Search and retrieve documents using the folder or the application group interface.
- Treat OnDemand folders as native entities in iees.
- Search synchronously and asynchronously in either the application group or folder mode.
- Retrieve entire OnDemand documents or document segments.
- Retrieve the logical view data of a given OnDemand document.
- Retrieve the resource group for a given OnDemand document.
- Retrieve annotation data for a given OnDemand document.
- Create and modify annotations.

Restriction: OnDemand does not support DB2 Text Information Extender and QBIC search or Combined query.

Representing OnDemand servers and documents

You represent a Content Manager OnDemand content server using a DKDatastoreOD in an Information Integrator for Content 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 DB2 Information Integrator for Content 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 DB2 Information Integrator for ContentVersion 8.3, DKViews, DKFixedView, and DKResource replace DKViewDataOD, DKFixedViewDataOD, and DKResouceGrpOD repsectively.

Connecting to and disconnecting from the OnDemand server

To log into an OnDemand content server, pass in the server name (for example, ODServer.mycompany.com), user ID, and password through the connect method.

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
System.out.println("connecting to datastore ...");
dsOD.connect(ODServer, UserID, Password, "");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();
cout << "connecting to datastore ..." << endl;
dsOD->connect(ODServer, UserID, Password, "");
```

Use the disconnect method to log out of the OnDemand server.

Java

```
System.out.println("disconnecting from the datastore ...");
dsOD.disconnect();
dsOD.destroy(); // Finished with the datastore
```

C++

```
cout << "disconnecting from the datastore ..." << endl;
dsOD->disconnect();
delete dsOD;
```

Listing information on OnDemand

You can list application groups and folders for OnDemand servers.

Listing application groups

You can list application groups in OnDemand using the `listEntities()` method of `DKDatastoreOD`. The following example illustrates how to use this method.

Java

```
...
pCol = (DKSequentialCollection) dsOD.listEntities(); //get application groups
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    agDef = (DKAppGrpDefOD)pIter.next();
    strAppGrp = agDef.getName();
    System.out.println("  app grp name[" + i + "]: " + strAppGrp);
    System.out.println("  show attributes for " + strAppGrp + " app grp - ");
    ...
}
```

C++

```
// ----- Show the application groups
// ----- First get the groups
pCol = (DKSequentialCollection*)dsOD.listEntities();
pIter = pCol->createIterator();
int i = 0;
// ----- Process the list
while (pIter->more() == TRUE)
{
    i++;
    agDef = (DKAppGrpDefOD*)((void*)(*pIter->next()));
    strAppGrp = agDef->getName();
    cout << "  app group name[" << i << "]: ==>" << strAppGrp << endl;
    . . .
}
```

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

Java

```
...
pCol2 = (DKSequentialCollection) dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2.createIterator();
j = 0;

while (pIter2.more() == true)
{
    j++;
    attrDef = (DKFieldDefOD)pIter2.next();
    System.out.println("    Attribute name[" + j + "]: " + attrDef.getName());
    System.out.println("        datastoreType: " + attrDef.datastoreType());
    System.out.println("        attributeOf: " + attrDef.getEntityName());
    System.out.println("        type: " + attrDef.getType());
    System.out.println("        size: " + attrDef.getSize());
    System.out.println("        id: " + attrDef.getId());
    System.out.println("        nullable: " + attrDef.isNullable());
    System.out.println("        precision: " + attrDef.getPrecision());
    System.out.println("        scale: " + attrDef.getScale());
    System.out.println("        stringType: " + attrDef.getStringType());
}

System.out.println("    " + j + " attribute(s) listed for " +
                    strAppGrp + " app grp\n");
...
```

C++

```
// ----- Get the attributes for each of the entities(application groups)
pCol2 = (DKSequentialCollection*)dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2->createIterator();
int j = 0;
// ----- List the attributes
while (pIter2->more() == TRUE)
{
    j++;
    attrDef = (DKFieldDefOD*)(void*)(*pIter2->next());
    cout << "attribute name[" << j << "]: ==>" << attrDef->getName() << endl;
    cout << "        datastore type: " << attrDef->datastoreType() << endl;
    cout << "        attribute of: " << attrDef->getEntityName() << endl;
    cout << "        type: " << attrDef->getType() << endl;
    cout << "        size: " << attrDef->getSize() << endl;
    cout << "        ID: " << attrDef->getId() << endl;
    cout << "        precision: " << attrDef->getPrecision() << endl;
    cout << "        scale: " << attrDef->getScale() << endl;
    cout << "        stringType: " << attrDef->getStringType() << endl;
    cout << "        nULLable: " << attrDef->isNullable() << endl;
    cout << "        queryable: " << attrDef->isQueryable() << endl;
    cout << "        updatable: " << attrDef->isUpdatable() << endl;
    // ----- Clean up the attribute
    delete attrDef;
}
cout << "    " << j << " attribute(s) listed for the " << strAppGrp
    << " app group\n" << endl;
// ----- Clean up the iterators and collections
if ( pIter2 )
    delete pIter2;
if ( pCol2 )
    delete pCol2;
...
```

Listing OnDemand folders

To get a list of folders in an OnDemand content server, you use the `listSearchTemplates()` function.

Java

```
...
dsDef = (DKDatastoreDefOD)dsOD.datastoreDef();
pCol = (DKSequentialCollection) dsDef.listSearchTemplates();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    folderName = (String)pIter.next();
    ....    // Process the folder as appropriate
}
dsOD.disconnect();
dsOD.destroy();
```

C++

```
. . .
// ----- List the folders
dsDef = (DKDatastoreDefOD*)dsOD.datastoreDef();
pCol = (DKSequentialCollection*)dsDef->listSearchTemplates();
pIter = pCol->createIterator();
i = 0;
// ----- Process the list of folders
while (pIter->more() == TRUE)
{
    i++;
    folderName = (DKString)(*pIter->next());
    cout << "folder name [" << i << "] - " << folderName << endl;
}
// ----- Disconnect
dsOD.disconnect();
. . .
```

Retrieving an OnDemand document

In an OnDemand server, you can retrieve documents. You can also display documents with their parts and attributes.

Searching for a particular document

The following example searches against an application group (OnDemand Publications) in the OnDemand content server.

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
String appgrp = "OnDemand Publications";
String SQLcmd = "where bookname LIKE 'A%'";

DKNVPair[] parms = new DKNVPair[3];
parms[0] = new DKNVPair("APPL_GROUP", appgrp);
parms[1] = new DKNVPair("MAX_RESULTS", new String(Integer.toString(5)));
parms[2] = new DKNVPair("CONTENT", new String("ATTRONLY"));

System.out.println("executing query");
dkResultSetCursor pCur = dsOD.execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
System.out.println("datastore executed query");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();

cout << "connecting to datastore ..." << endl;
dsOD->connect(ODServer, UserID, Password, "");

DKString appgrp = "OnDemand Publications";
DKString SQLcmd = "where bookname LIKE 'A%'";

DKNVPair parms[4];
parms[0] = DKNVPair("APPL_GROUP", appgrp);
parms[1] = DKNVPair("MAX_RESULTS", DKString(5));
parms[2] = DKNVPair("CONTENT", DKString("ATTRONLY"));
parms[3] = DKNVPair( DK_CM_PARM_END, DKAny((long)0) );

cout << "executing query" << endl;
dkResultSetCursor* pCur = dsOD->execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
cout << "datastore executed query" << endl;
if (pCur != 0)
    delete pCur;
```

The following example searches against an application group (OnDemand Publications), and retrieves the documents returned.

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
String appgrp = "OnDemand Publications";
String SQLcmd = "where bookname LIKE 'A%'";

DKNVPair[] parms = new DKNVPair[3];
parms[0] = new DKNVPair("APPL_GROUP", appgrp);
parms[1] = new DKNVPair("MAX_RESULTS", new String(Integer.toString(5)));
parms[2] = new DKNVPair("CONTENT", new String("ATTRONLY"));

System.out.println("executing query");
dkResultSetCursor pCur = dsOD.execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
System.out.println("datastore executed query");

while (pCur.isValid())
{
    DKDDO p = pCur.fetchNext();
    if (p != null)
    {
        String idstr = ((DKPid)p.getPidObject()).pidString();
        System.out.println(" pidString : " + idstr);
        DKPid pid = new DKPid (idstr);
        DKDDO ddoold = p;
        short id, docType = 0;
        if ((id = ddoold.propertyId(DK_CM_PROPERTY_ITEM_TYPE)) > 0)
            docType = ((Short)ddoold.getProperty(id)).shortValue();
        if (docType == DK_CM_DOCUMENT)
        {
            System.out.println("create a new DDO with a cloned pid to retrieve!");
            p = dsOD.createDDO(ddoold.getObjectType(), DK_CM_DOCUMENT);
            p.setPidObject(pid);
            try
            {
                dsOD.retrieveObject((dkDataObject)p);
            }
            catch (DKException exc)
            {
                System.out.println("Exception name " + exc.name());
                System.out.println("Exception message " + exc.getMessage());
                System.out.println("Exception error code " + exc.errorCode());
                System.out.println("Exception error state " + exc.errorState());
                exc.printStackTrace();
            }
        }
    }
}
pCur.destroy(); // Finished with the cursor
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();
DKString appgrp = "OnDemand Publications";
DKString SQLcmd = "where bookname LIKE 'A%'";

DKNVPair parms[4];
parms[0] = DKNVPair("APPL_GROUP", appgrp);
parms[1] = DKNVPair("MAX_RESULTS", DKString(5));
parms[2] = DKNVPair("CONTENT", DKString("ATTRONLY"));
parms[3] = DKNVPair( DK_CM_PARM_END, DKAny((long)0) );

cout << "executing query" << endl;
dkResultSetCursor* pCur = dsOD->execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
cout << "datastore executed query" << endl;

if (pCur != 0)
{
    while (pCur->isValid())
    {
        DKDDO* p = pCur->fetchNext();
        DKDDO* ddoold = p;
        DKString pidStr = ((DKPid*)ddoold->getPidObject())->pidString();
        DKPid* pid = new DKPid(pidStr);
        short id, docType = 0;
        DKAny a;
        if ((id = ddoold->propertyId(DK_CM_PROPERTY_ITEM_TYPE)) > 0)
        {
            a = ddoold->getProperty(id);
            if (a.typeCode() == DKAny::tc_ushort)
                docType = (short)(USHORT)a;
            else
                docType = a;
        }
        if (docType == DK_CM_DOCUMENT)
        {
            cout << "create the DDO from the pidstring..." << endl;
            p = dsOD->createDDO(ddoold->getObjectType(), DK_CM_DOCUMENT);
            p->setPidObject(pid);

            dsOD->retrieveObject((dkDataObject*)p);
        }
        delete pid;
        delete ddoold;
    }
    delete pCur;
}
```

Displaying documents and their parts and attributes

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

Java

```
//-----For each data item, get the attributes
//-----numDataItems is the number of data items
for (j = 1; j <= numDataItems; j++)
{
    a = p.getData(j)
    strDataName = p.getDataName(j);
    System.out.println("    " + j + ". Attribute Name: " + strDataName );
    System.out.println("        type: " + p.getDataPropertyByName
        (j,DK_PROPERTY_TYPE));

    System.out.println("        nullable: " +
        p.getDataPropertyByName (j,DK_PROPERTY_NULLABLE));
    if (strDataName.equals(DKPARTS) == false &&
        strDataName.equals("DKResource") == false &&
        strDataName.equals("DKViews") == false &&
        strDataName.equals("DKLargeObject") == false &&
        strDataName.equals("DKFixedView") == false &&
        strDataName.equals("DKAnnotations") == false)
    {
        System.out.println("        attribute id: " +
            p.getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID));
    }
    //-----Check for the type of the attribute
    if (a instanceof String)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof Integer)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof Short)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof DKDate)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof DKTime)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof DKTimestamp)
    {
        System.out.println("        Attribute Value: " + a);
    }
    else if (a instanceof dkCollection)
    {
        System.out.println("        Attribute Value is collection");
        pCol = (dkCollection)a;
        pIter = pCol.createIterator();
        i = 0;
        while (pIter.more() == true)
        {
            i++;
            a = pIter.next();
            pDO = (dkDataObjectBase)a;
        }
    }
}

// continued...
```

Java (continued)

```
if (pD0.protocol() == DK_XD0)
{
    System.out.println("    dkXD0 object " + i + " in collection");
    pXD0 = (dkXD0)pD0;
    DKPidXD0 pid2 = pXD0.getPidObject();
    System.out.println("        XD0 pid string: " +
        pid2.pidString());
    //----- Retrieve and open instance handler for an XD0
    pXD0.retrieve();
    // pXD0.open();
}
}
}
else if (a != null)
{
    System.out.println("    Attribute Value: " + a.toString());
    if (strDataName.equals("DKResource") ||
        strDataName.equals("DKFixedView") ||
        strDataName.equals("DKLargeObject"))
    {
        pD0 = (dkDataObjectBase)a;

        if (pD0.protocol() == DK_XD0)
        {
            System.out.println("        dkXD0 object ");
            pXD0 = (dkXD0)pD0;
            DKPidXD0 pid2 = pXD0.getPidObject();
            System.out.println("            XD0 pid string: " +
                pid2.pidString());
            // Retrieve and open instance handler for an XD0
            pXD0.retrieve();
            // pXD0.open();
        }
    }
}
```

C++

```
DKDDO *p = 0;
DKAny a;

for (j = 1; j <= numDataItems; j++)
{
    a = p->getData(j);
    strDataName = p->getDataName(j);

    cout << " " << j << ". Attribute Name: " << strDataName << endl;
    cout<<"type: "<< p->getDataPropertyByName(j,DK_PROPERTY_TYPE)<<endl;
    cout << "nullable: "
        << p->getDataPropertyByName(j,DK_PROPERTY_NULLABLE) << endl;

    if (strDataName != DK_CM_DKPARTS    &&
        strDataName != "DKResource"    &&
        strDataName != "DKViews"       &&
        strDataName != "DKLargeObject" &&
        strDataName != "DKPermissions" &&
        strDataName != "DKFixedView"   &&
        strDataName != "DKAnnotations")
    {
        cout << "    attribute ID: "
            << p->getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID) << endl;
    }

    if (a.typeCode() == DKAny::tc_string)
    {
        DKString astring = a;
        cout << "    attribute Value (string): " << astring << endl;
    }
    else if . . .
    {
        // ----- Handle each of the other types
    }
    else if (a.typeCode() != DKAny::tc_null)
    {
        cout << "    Attribute Value (non NULL): " << a << endl;
        if (strDataName == "DKResource"    ||
            strDataName == "DKFixedView"   ||
            strDataName == "DKLargeObject")
        {
            pDO = (dkDataObjectBase*)a;
            if (pDO->protocol() == DK_XDO)
            {
                cout << "        dkXDO object " << endl;
                pXDO = (dkXDO*)pDO;
                pidXDO = (DKPidXDOOD*)pXDO->getPid();
                cout << "        XDO PID string: " << pidXDO->pidString() << endl;
                // ----- Retrieve and open instance handler for an XDO
                pXDO->retrieve();
            }
        }
    }
    else cout << " Attribute Value is NULL" << endl;
}
```

For the complete application, refer to TRetrieveOD.cpp in the samples directory.

Enabling the OnDemand folder mode

To enable the OnDemand folder mode, the string
ENTITY_TYPE=TEMPLATES

must be passed to the OnDemand connector as part of the connection string or the configuration string. An sample configuration string would look like this:

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

A sample connect string would look like this:

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();  
dsOD.connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");  
dsOD->connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

Asynchronous search

The OnDemand connector supports both federated and direct asynchronous searches. An asynchronous search does not tie up the main thread and allows the search to be cancelled at any time; press the **Stop Search** button on the Search Template Viewer to terminate the search. The max hits property on the Search Template Viewer limits the maximum number of results returned.

The OnDemand connector also supports synchronous and asynchronous searches in the application group mode from an AIX client.

If you use a search criterion such as WHERE userid LIKE '%', the resulting number of documents returned to the client can consume all available memory on the client's machine. By issuing an asynchronous search using the executeWithCallback() method, you can set the value for the maximum number of documents returned and cancel the search at any time.

You may also have to increase the default Java Virtual Machine (JVM) stack size if your result set is too large. The default stack size for each Java thread is 400k, which allows 3920 return items before the stack overflows. Increasing the JVM stack size to 800k doubles the capacity to 7840 items. If necessary, you can further increase the JVM stack size.

To raise the JVM stack size, use the Java command line option -oss followed by nnnK or nnM, where K stands for Kilobytes and M for Megabytes.

For examples of using asynchronous search, see the `TRetrieveWithCallbackOD`, `TRetrieveFolderWithCallbackOD` and `TCallbackOD` sample programs.

OnDemand folders as search templates

Three of the Information Integrator for Content visual JavaBeans, `CMBSearchTemplateList`, `CMBSearchTemplateViewer`, and `CMBSearchResultsView`, use Information Integrator for Content 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.setConnectionString("ENTITY_TYPE=TEMPLATES");
```

OnDemand folders as native entities

The OnDemand connector can also map OnDemand folders as native entities by specifying the connect string `"ENTITY_TYPE=TEMPLATES"`. Using OnDemand folders as entities can be useful in federated searches, where folder definitions may be easier to work with than OnDemand application groups, the default native entity for OnDemand.

For federated searches, specify the server definition in DB2 Information Integrator for Content administration. You can then define and map federated entities to the folders on the OnDemand server.

Create and modify annotations

When using the OnDemand viewer, which is launched by the `CMBDocumentViewer` bean, you can create, modify, and delete annotations for OnDemand documents by using the `CMBDataManagement` bean and the associated `CMBAnnotation` class.

Tracing

You can trace events with the OnDemand connector. To enable the connector Java API trace, place the trace INI file (`cmbodtrace.ini` for Java or `cmbodctrace.inifor C++`), in the root of the C drive. For AIX, place this file in `/opt/IBM/db2cmv8`. For Solaris, place this file in `/opt/IBMcmb/cmgmt`. For Linux, also place this file in `/opt/IBM/db2cmv8`.

The default output directory for trace files is `C:\Ctrace`. To write the trace information elsewhere, edit the trace INI file. For AIX, note that the file names must be all lower case.

Make sure that the path name specified in the trace file is valid and that the line containing `CMBODTRACEDIR` is not preceded with a `#` sign. Here are the sample trace INI files:

Java

```
#=====
# This is a java property file - not a real INI file!
# *****#
# For windows systems, make sure to use TWO BACK SLASH CHARACTERS (\\)
# to separate the directory names!!! =====
# *****#
#
# ***** On windows systems, this file must be located in c:\ *****
#
# OD Trace File Directory Name property - CMBODTRACEDIR
#
# The CMBODTRACEDIR property defines the directory where the trace files
# will be written to. If the directory name does not exist, it will be
# created.
#
# Please make sure that the directory names are separated by two back slash
# characters to avoid undesirable results.
#
# Please make sure the path name does not point to an existing file name.
# Otherwise, no trace files will be created.
#
# The trace output directory name can be changed to point to a drive
# where more space is available. But it is recommended not to change the
# trace output directory name in the middle of an active trace session.
#
# CMBODTRACESCOPE controls how much trace information to generate.
#
# CMBODTRACESCOPE=ENTRY_EXIT_JNI_ONLY
# Trace the entry & exit points in JNI only. Produce the least amount
# of trace.
#
# CMBODTRACESCOPE=ENTRY_EXIT_ONLY
# Trace the entry and exit points in Java methods and JNI functions.
#
# CMBODTRACESCOPE=JNI_ONLY
# Full trace for the JNI functions only.
#
# If CMBODTRACESCOPE is missing, or set to anything else,
# a full trace will be taken.
#
# To disable the trace, add a leading # character in column 1.
#
# AIX: change the following line to CMBODTRACEDIR=/usr/lpp/cmb/cmgmt/trace
# Sun: change the following line to CMBODTRACEDIR=/opt/IBMcmb/cmgmt/trace
CMBODTRACEDIR=c:\\trace
```


C++

```
#=====
# OnDemand Trace INI file
#
# OnDemand Trace File Directory Name key - CMBODTRACEDIR
#
# The CMBODTRACEDIR key defines the directory where the trace files will
# be written to. If the directory name does not exist, it will be created.
#
# Please make sure the path name does not point to an existing file name.
# Otherwise, no trace files will be created.
#
# The trace output directory name can be changed to point to a drive
# where more space is available. But it is recommended not to change
# the trace output directory name in the middle of an active trace
# session.
#
# CMBODTRACESCOPE controls how much trace information to generate.
#
# CMBODTRACESCOPE=ENTRY_EXIT_ONLY
# Trace only the entry and exit of all C++ methods and functions.
#
# If CMBODTRACESCOPE is missing, or set to anything else, a full trace
# is taken.
#
# To disable the trace, add a leading # character in column 1 on
# the CMBODTRACEDIR line.
#
[ODCTRACE]
# For AIX: change next line to CMBODTRACEDIR=/usr/lpp/cmb/cmgt/ctrace
CMBODTRACEDIR=D:\Ctrace
#CMBODTRACESCOPE=ENTRY_EXIT_ONLY
```

Working with Content Manager ImagePlus for OS/390

Information Integrator for Content APIs support the following features when working with Content Manager ImagePlus for OS/390 content servers:

- Connecting and disconnecting from one or more Content Manager ImagePlus for OS/390 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: Content Manager ImagePlus for OS/390 does not support:

- DB2 Text Information Extender and QBIC search
- Combined query
- Workbasket and workflow

Listing entities and attributes

After creating a content server for an Content Manager ImagePlus for OS/390 content server as a DKDatastoreIP object and connecting, you can check the entities

and attributes for the content server. The following example lists all of the entities for an Content Manager ImagePlus for OS/390 content server:

Java

```
// ----- After creating a datastore and connecting
//          dsIP is a DKDatastoreIP object
DKEntityDefIP entDef = null;
DKAttrDefIP attrDef = null;

DKSequentialCollection pCol = (DKSequentialCollection)dsIP.listEntities();
dkIterator pIter = null;

if ( pCol == null )
{
    // ----- Handle if the collection of entities is null
}
else
{
    ... // ----- Process as appropriate
```

The complete sample application from which this example was taken (TListCatalogIP.java) is available in the samples directory.

C++

```
// List entities...
DKEntityDefIP* docDef = 0;
DKAttrDefIP* attrDef = 0;

cout << "---List entities---" << endl;
DKSequentialCollection* pCol = (DKSequentialCollection*)(dsIP.listEntities());
dkIterator* pIter = 0;

if ( pCol == 0 )
{
    cout << "collection of entities is null!" << endl;
}
else
{
    ...
```

The complete sample application from which this example was taken (TListCatalogIP.cpp) is available in the samples directory.

The following example lists all of the attributes associated with each entity using the `getAttr` and `listAttrNames` functions of `DKEntityDefIP`.

Java

```
// ----- List attributes using listAttrNames and getAttr methods

pIter = pCol.createIterator();
while (pIter.more())
{
    // ----- Iterate over the each entity
    entDef = (DKEntityDefIP)pIter.next();
    System.out.println(" Entity type      : " + entDef.getType() );
    System.out.println(" Entity type name: " + entDef.getName() );

    // ----- Get a list of attributes for the entity
    String[] attrNames = entDef.listAttrNames();
    int count = attrNames.length;
    for (int i = 0; i < count; i++)
    {
        attrDef = (DKAttrDefIP)entDef.getAttr( attrNames[i] );
        System.out.println("  Attr name      : " + attrDef.getName() );
        System.out.println("  Attr id        : " + attrDef.getId() );
        System.out.println("  Entity name    : " + attrDef.getEntityName() );
        System.out.println("  Datastore name: " + attrDef.datastoreName() );
        System.out.println("  Attr type      : " + attrDef.getType() );
        System.out.println("  Attr restrict  : " + attrDef.getStringType() );
        System.out.println("  Attr min val   : " + attrDef.getMin() );
        System.out.println("  Attr max val   : " + attrDef.getMax() );
        System.out.println("  Attr display   : " + attrDef.getSize() );
        System.out.println("  Attr precision: " + attrDef.getPrecision() );
        System.out.println("  Attr scale     : " + attrDef.getScale() );
        System.out.println("  Attr update ?  : " + attrDef.isUpdatable() );
        System.out.println("  Attr nullable ? : " + attrDef.isNullable() );
        System.out.println("  Attr queryable? : " + attrDef.isQueryable() );
        System.out.println(" ");
    }
}
```

C++

```
// Method 1:
cout << "List attributes using listAttrNames and getAttr functions" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
    docDef = (DKEntityDefIP*)(pIter->next()->value());
    cout << " Document type      : " << docDef->getType() << endl;
    cout << " Document type name: " << docDef->getName() << endl;

    long tmpCount;
    DKString* attrNames;

    // Upon return, tmpCount contains the number of elements in the list.
    attrNames = docDef->listAttrNames(tmpCount);
    for (int i=0; i<tmpCount; i++)
    {
        cout << "  Attr name before lookup " << attrNames[i] << endl;
        attrDef = (DKAttrDefIP*)(docDef->getAttr(attrNames[i]));
        cout << "  Attr name [" << i << "] : " << attrDef->getName() << endl;
        cout << "  Attr id      : " << attrDef->getId() << endl;
        cout << "  Entity name  : " << attrDef->getEntityName() << endl;
        cout << "  Datastore name: " << attrDef->datastoreName() << endl;
        cout << "  Attr type    : " << attrDef->getType() << endl;
        cout << "  Attr restrict : " << attrDef->getStringType() << endl;
        cout << "  Attr min val  : " << attrDef->getMin() << endl;
        cout << "  Attr max val  : " << attrDef->getMax() << endl;
        cout << "  Attr display  : " << attrDef->getSize() << endl;
        cout << "  Attr precision: " << attrDef->getPrecision() << endl;
        cout << "  Attr scale    : " << attrDef->getScale() << endl;
        cout << "  Attr update   ? " << attrDef->isUpdatable() << endl;
        cout << "  Attr nullable ? " << attrDef->isNullable() << endl;
        cout << "  Attr queryable? " << attrDef->isQueryable() << endl;
        cout << "" << endl;
        delete attrDef;
    } // end for

    delete [] attrNames;

} // end while
delete pIter;
```

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

Java

```
// --- List attributes using listEntityAttrs method
pIter = pCol.createIterator();
while (pIter.more())
{
    entDef = (DKEntityDefIP)pIter.next();
    System.out.println(" Entity type      : " + entDef.getType() );
    System.out.println(" Entity type name: " + entDef.getName() );

    DKSequentialCollection pAttrCol =
        (DKSequentialCollection)dsIP.listEntityAttrs(entDef.getName());
    if ( pAttrCol == null )
    {
        // ----- Handle if the collection of attributes is null
    }
    else
    {
        dkIterator pAttrIter = pAttrCol.createIterator();
        while (pAttrIter.more())
        {
            attrDef = (DKAttrDefIP)pAttrIter.next();
            System.out.println("  Attr name      : " + attrDef.getName() );
            System.out.println("  Attr id        : " + attrDef.getId() );
            System.out.println("  Entity name    : " + attrDef.getEntityName() );
            System.out.println("  Datastore name: " + attrDef.datastoreName() );
            System.out.println("  Attr type      : " + attrDef.getType() );
            System.out.println("  Attr restrict  : " + attrDef.getStringType() );
            System.out.println("  Attr min val   : " + attrDef.getMin() );
            System.out.println("  Attr max val   : " + attrDef.getMax() );
            System.out.println("  Attr display   : " + attrDef.getSize() );
            System.out.println("  Attr precision: " + attrDef.getPrecision() );
            System.out.println("  Attr scale     : " + attrDef.getScale() );
            System.out.println("  Attr update    ? " + attrDef.isUpdatable() );
            System.out.println("  Attr nullable ? " + attrDef.isNullable() );
            System.out.println("  Attr queryable? " + attrDef.isQueryable() );
            System.out.println(" ");
        }
    }
}
```

C++

```
// Method 2:
cout << "----List attributes using listEntityAttrs function----" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
    docDef=(DKEntityDefIP*)(pIter->next()->value()); //iterator returns DKAny*
    cout << " Document type      : " << docDef->getType() << endl;
    cout << " Document type name: " << docDef->getName() << endl;
    DKSequentialCollection* pAttrCol = (DKSequentialCollection*)
                                        (dsIP.listEntityAttrs(docDef->getName()));

    if ( pAttrCol == 0 )
    {
        cout << "collection of entity attrs is null for entity "
              << docDef->getName()
              << endl;
    }
    else
    {
        int i=0;
        dkIterator* pAttrIter = pAttrCol->createIterator();
        while (pAttrIter->more())
        {
            i++;
            // ----- The iterator returns a pointer to DKAny
            attrDef = (DKAttrDefIP*)(pAttrIter->next()->value());
            cout << "  Attr name [" << i << "] : " << attrDef->getName() << endl;
            cout << "  Attr id      : " << attrDef->getId() << endl;
            cout << "  Entity name  : " << attrDef->getEntityName() << endl;
            cout << "  Datastore name: " << attrDef->datastoreName() << endl;
            cout << "  Attr type    : " << attrDef->getType() << endl;
            cout << "  Attr restrict : " << attrDef->getStringType() << endl;
            cout << "  Attr min val  : " << attrDef->getMin() << endl;
            cout << "  Attr max val  : " << attrDef->getMax() << endl;
            cout << "  Attr display  : " << attrDef->getSize() << endl;
            cout << "  Attr precision: " << attrDef->getPrecision() << endl;
            cout << "  Attr scale    : " << attrDef->getScale() << endl;
            cout << "  Attr update   ? " << attrDef->isUpdatable() << endl;
            cout << "  Attr nullable ? " << attrDef->isNullable() << endl;
            cout << "  Attr queryable? " << attrDef->isQueryable() << endl;
            cout << "" << endl;
            delete attrDef;
        } // end while
        delete pAttrIter;
    }
    delete pAttrCol;
    delete docDef;
} // end while
delete pIter;
}
```

ImagePlus for OS/390 query syntax

The following example shows the query syntax for ImagePlus for OS/390:

Java

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

C++

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

The query uses the following parameters:

search_expression

Each search expression consists of one or more search criteria. You can only use the boolean operator AND between search criteria.

The search criteria has the form:

{attr_name | mapped_attr_name} operator literal

where:

attr_name

Name of the entity attribute on which to base the search.

mapped_attr_name

Attribute name mapped with the attribute on which to base the search.

operator

All attributes support equality (==). For attributes of type DATE, you can use the following additional operators:

- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to

literal

A literal. For numeric attributes, do not use quotation marks ("), for example:

```
FolderType == 9
```

For date, time, and timestamp attributes, quotation marks or apostrophes (') are not necessary, but are tolerated, for example:

```
ReceiveDate == 1999-03-08  
ReceiveDate == '1999-03-08'
```

For string attributes, quotation marks or apostrophes (') are not necessary, but are tolerated. If the string contains an apostrophe ('), the string must be specified using two apostrophes, for example for a value of Folder'1:

```
FolderId == 'Folder''1'
```

entity_name

Name of the entity to be searched.

mapped_entity_name

Entity name mapped to the entity to be searched.

maximum_results

Maximum number of results to return.

The option keywords are:

CONTENT Controls the amount of information returned in the results

YES (default)

Sets the PIDs, attributes and their values for a document or folder. If there are parts in a document, the XDO PIDs are set. If there are documents in a folder, the document PIDs are set.

NO Only sets the document or folder PIDs.

ATTRONLY

Sets only the PIDs, attributes and their values for a document or folder.

PENDING Controls whether to include pending documents that do not have any parts. This option only applies when ENTITY is set to DOCUMENT or to an entity mapped to DOCUMENT.

YES Includes pending documents in the results.

NO (default)

Does not include pending documents in the results.

Working with DB2 Content Manager for AS/400

The API classes provided for DB2 Content Manager for AS/400 (VisualInfo for AS/400) are similar to those provided for DB2 Content Manager.

Restriction: DB2 Content Manager for AS/400 does not support:

- DB2 Text Information Extender and QBIC search
- Combined query
- Workbasket and workflow

Listing entities (index classes) and attributes

You represent a DB2 Content Manager for AS/400 content server as a DKDatastoreV4. After creating the content server and connecting to it, you can list the entities (index classes) and attributes for the DB2 Content Manager for AS/400 server (see the example).

Java

```
// ----- After creating a datastore (dsV4) and connecting, get index classes
pCol = (DKSequentialCollection) dsV4.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    icDef = (DKIndexClassDefV4)pIter.next();
    strIndexClass = icDef.getName();
    ... // ---- Process the index classes as appropriate
    // ----- Get the attributes
    pCol2 = (DKSequentialCollection) dsV4.listEntityAttrs(strIndexClass);
    pIter2 = pCol2.createIterator();
    j = 0;

    while (pIter2.more() == true)
    {
        j++;
        attrDef = (DKAttrDefV4)pIter2.next();
        ... // ----- Process the attributes
    }
}

dsV4.disconnect();
dsV4.destroy();
```

The complete sample application from which this example was taken (TListCatalogV4.java) is available in the samples directory.

C++

```
cout << "list index class(es)..." << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsV4.listSchema());
pIter = pCol->createIterator();
i = 0;

while (pIter->more() == TRUE)
{
    i++;
    a = (*pIter->next());
    strIndexClass = a;
    cout << "index class name [" << i << "] - " << strIndexClass << endl;
    cout << " list attribute(s) for " << strIndexClass << " index class:" << endl;
    pCol2 =

(DKSequentialCollection*)((dkCollection*)dsV4.listSchemaAttributes(strIndexClass));
    pIter2 = pCol2->createIterator();
    j = 0;

    while (pIter2->more() == TRUE)
    {
        j++;
        pA = pIter2->next();
        pDef = (DKAttributeDef*) pA->value();
        cout << "    Attribute name [" << j << "] - " << pDef->name << endl;
        cout << "    datastoreType - " << pDef->datastoreType << endl;
        cout << "    attributeOf - " << pDef->attributeOf << endl;
        cout << "    type - " << pDef->type << endl;
        cout << "    size - " << pDef->size << endl;
        cout << "    id - " << pDef->id << endl;
        cout << "    nullable - " << pDef->nullable << endl;
        cout << "    precision - " << pDef->precision << endl;
        cout << "    scale - " << pDef->scale << endl;
        cout << "    string type - " << pDef->stringType << endl;
    }

    cout << " " << j << " attribute(s) listed for "
        << strIndexClass << " index class" << endl;
    pCol2->apply(deletedDKAttributeDef);
    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 samples directory.

Running a query

The following example runs a query in DB2 Content Manager for AS/400, and processes the results.

Java

```
// ----- After creating a datastore (dsV4) and connecting, build the
//          query and parameters and execute it
pCur = dsV4.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
...

if (pCur == null)
{
    // ---- Handle if the cursor is null
}

while (pCur.isValid())
{
    p = pCur.fetchNext();
    if (p != null)
    {
        cnt++;
        i = pCur.getPosition();
        System.out.println("\n====> Item " + i + " <====");
        numDataItems = p.dataCount();
        DKPid pid = p.getPid();
        System.out.println("  pid string: " + pid.pidString());
        k = p.propertyId(DK_CM_PROPERTY_ITEM_TYPE);

        if (k > 0)
        {
            Short sVal = (Short)p.getProperty(k);
            j = sVal.shortValue();
            switch (j)
            {
                case DK_CM_DOCUMENT :
                {
                    ... // Handle if the item is a document ";
                    break;
                }
                case DK_CM_FOLDER :
                {
                    ... // Handle if the item is a folder
                    break;
                }
            }
        }
    }
}

for (j = 1; j <= numDataItems; j++)
{
    a = p.getData(j);
    strDataName = p.getDataName(j);
    ... // Process the attributes as appropriate
    if (strDataName.equals(DKPARTS) == false
        && strDataName.equals(DKFOLDER) == false)
    {
        System.out.println("      attribute id: "
            + p.getDataPropertyByName(j,DK_CM_PROPERTY_ATTRIBUTE_ID));
    }
}
// continued...
```

Java (continued)

```
if (a instanceof String)
{
    System.out.println("        Attribute Value: " + a);
}
else if (a instanceof Integer)
    ... // ---- Handle each type for attribute {
else if (a instanceof dkCollection)
{
    // ---- Handle if attribute value is a collection
    pCol = (dkCollection)a;
    pIter = pCol.createIterator();
    i = 0;
    while (pIter.more() == true)
    {
        i++;
        a = pIter.next();
        pDO = (dkDataObjectBase)a;

        if (pDO.protocol() == DK_CM_PDDO)
        {
            // Process a DDO
            pDDO = (DKDDO)pDO;
            ...
        }
        else if (pDO.protocol() == DK_CM_XDO)
        {
            // Process an XDO
            pXDO = (dkXDO)pDO;
            DKPidXDO pid2 = pXDO.getPid();
            ...
        }
    }
}
else if (a != null)
{
    // Process the attribute
}
else ... // Handle if the attribute is null
}
}
pCur.destroy(); // Delete the cursor when you're done
```

The complete sample application from which this example was taken (TExecuteV4.java) is available in the samples directory.

C++

```
cout << "executing query..." << endl;
...
pCur = dsV4.execute(cmd);
cout << "  query executed" << endl;
...
cout << "\n..... Displaying query results ..... \n\n";

...
while (pCur->isValid())
{
    p = pCur->fetchNext();

    if (p != 0)
    {
        cout << "======" << "Item " << cnt << " <======" << endl;
        numDataItems = p->dataCount();
        pid = p->getPid();
        cout << "  Pid String: " << pid.pidString() << endl;
        k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);

        if (k > 0)
        {
            a = p->getProperty(k);
            val = a;
            cout << "  *****" << endl;

            switch (val)
            {
                case DK_CM_DOCUMENT :
                {
                    cout << "  Item is a document " << endl;
                    break;
                }
                case DK_CM_FOLDER :
                {
                    cout << "  Item is a folder " << endl;
                    break;
                }
            }

            cout << "  *****" << endl;
        }

        cout << "  Number of Data Items: " << numDataItems << endl;

        for (j = 1; j <= numDataItems; j++)
        {
            a = p->getData(j);
            strDataName = p->getDataName(j);

            switch (a.typeCode())
            {
                case DKAny::tc_string :
                {
                    strData = a;
                    cout << "  attribute name: " << strDataName
                        << ", value: " << strData << endl;
                    break;
                }
            }
        }
    }
}

// continued...
```

C++ (continued)

```
case DKAny::tc_long :
{
    lVal = a;
    cout << " attribute name: " << strDataName
        << ", value: " << lVal << endl;
    break;
}

case DKAny::tc_null :
{
    cout<<" attribute name: "<<strDataName<<", value: NULL "<< endl;
    break;
}

case DKAny::tc_collection :
{
    pdCol = a;
    cout<<strDataName<<" collection name: "<<strDataName << endl;
    cout<<"-----"<<endl;
    pdIter = pdCol->createIterator();
    ushort b = 0;

    while (pdIter->more() == TRUE)
    {
        b++;
        cout << " -----" << endl;
        a = *(pdIter->next());
        pDOBase = a;

        if (pDOBase->protocol() == DK_PDDO)
        {
            pDDO = (DKDDO*)pDOBase;
            cout << " DKDDO object " << b << " in " << strDataName
                << " collection " << endl;
            k = pDDO->propertyId(DK_CM_PROPERTY_ITEM_TYPE);

            if (k > 0)
            {
                a = pDDO->getProperty(k);
                val = a;
                cout << " *****" << endl;

                switch (val)
                {
                    case DK_CM_DOCUMENT :
                    {
                        cout << " Item is a document " << endl;
                        break;
                    }
                    case DK_CM_FOLDER :
                    {
                        cout << " Item is a folder " << endl;
                        break;
                    }
                }
                cout << " *****" << endl;
            }
        }
    }
}

// continued...
```

C++ (continued)

```
        else if (pDObase->protocol() == DK_XDO)
        {
            pXDO = (dkXDO*)pDObase;
            cout << " dkXDO object " << b << " in " << strDataName
                << " collection " << endl;

            }
        }

        if (pdIter != 0)
        {
            delete pdIter;
        }

        if (b == 0)
        {
            cout << strDataName << " collection has no elements " << endl;
        }

        cout << " -----" << endl;
        break;
    }

    default:
        cout << "Type is not supported\n";
    }

    cout<<"type: "<< p->getDataPropertyByName(j,DK_CM_PROPERTY_TYPE)<<endl;
    cout<<"nullable: "<< p->getDataPropertyByName(j,DK_CM_PROPERTY_NULLABLE)
        << endl;

    if (strDataName != DKPARTS && strDataName != DKFOLDER)
    {
        cout << " attribute id: "
            << p->getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID) << endl;
    }
    }
    cnt++;
    delete p;
}
}
cout << "Total Item count is " << cnt-1 << endl;

if (pCur != 0)
    delete pCur;
```

The complete sample application from which this application was taken (TExecuteV4.cpp) is available in the samples directory.

Running a parametric query

The following example runs a parametric query.

Java

```
// ----- Create the query string and the query object
String cmd = "SEARCH=(INDEX_CLASS=V4DEMO)";
pQry = dsV4.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Run the query
pQry.execute(parms);

System.out.println("number of query results = " + pQry.numberOfResults());

// ----- Processing the query results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
...
```

The complete sample application from which this example was taken (TSamplePQryV4.java) is available in the samples directory.

C++

```
cout << "query string: " << cmd << endl;
cout << "creating query..." << endl;
pQry = dsV4.createQuery(cmd);
cout << "executing query..." << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "getting query results..." << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsV4.disconnect();
cout << "datastore disconnected" << endl;
delete pQry;
delete pResults;
```

The complete sample application from which this application was taken (TSamplePQryV4.cpp) is available in the samples directory.

Working with Domino.Doc

Domino.Doc is the Lotus Domino solution for organizing, managing, and storing business documents, and making them accessible within and outside of a business.

Domino.Doc supports the open document management API (ODMA), so that you can create, save, and retrieve documents using ODMA-enabled applications. ODMA connects to a Domino.Doc server using an HTTP or Lotus Notes protocol.

Domino.Doc includes the following features:

- Connecting and disconnecting from one or more Domino.Doc servers.
- Ability to search binders.
- Ability to retrieve documents.
- ODMA compliance so that users can work in familiar applications.

Restriction: Domino.Doc does not support:

- Add, update, and delete document methods.
- DB2 Text Information Extender and QBIC search.
- Combined query.
- Workbasket and workflow.

When using the API to work with a Domino.Doc object, you must build an expression to return the objects. This section describes the design of the Domino.Doc API, how objects fit into the hierarchy, and how to build the expression. Figure 19 shows the relationship between the Domino.Doc object model and its components.

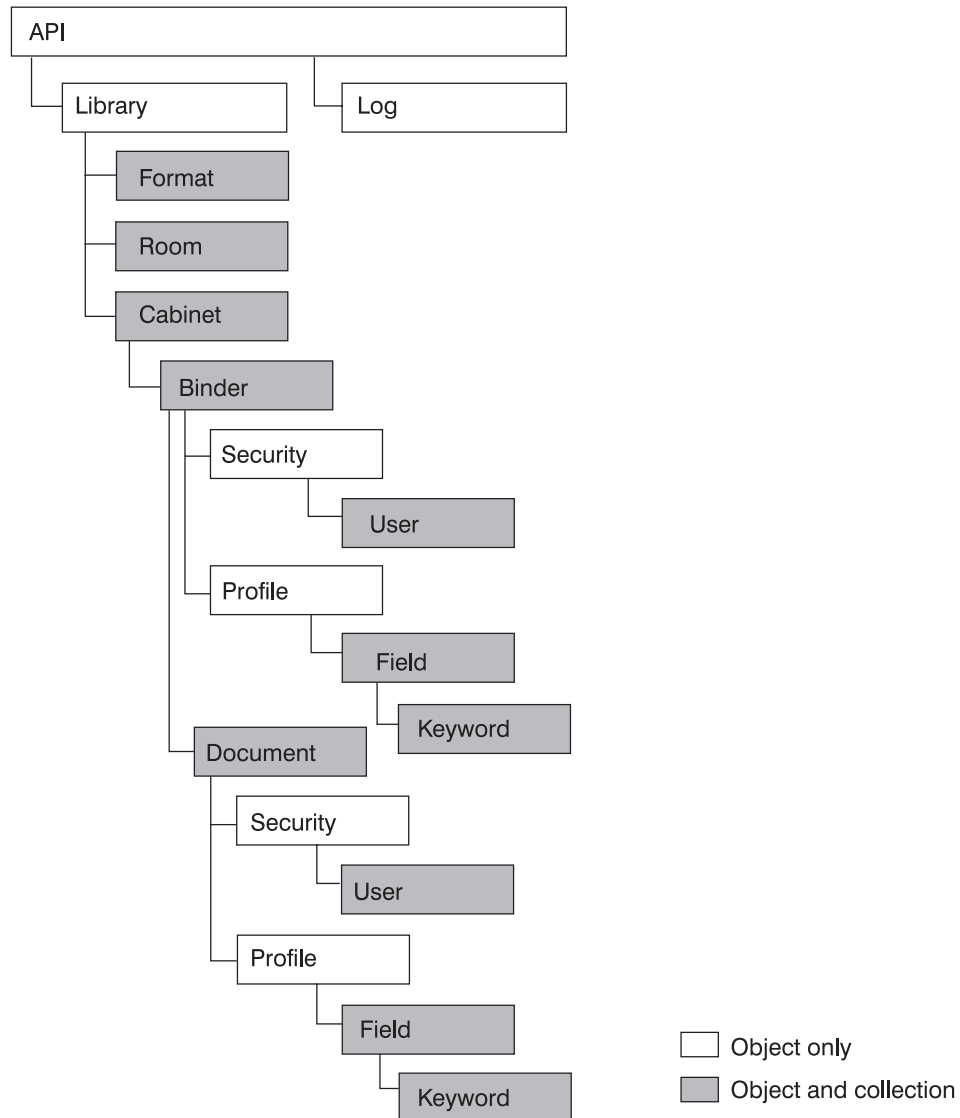


Figure 19. Domino.Doc object model

The elements contained in Domino.Doc (and the APIs to represent them) are arranged such that:

- The library contains rooms (DKRoomDefDD objects) and cabinets (DKCabinetDefDD objects).
- Each cabinet contains binders (DKBinderDefDD object).

- Each binder contains a profile (DKAttrProfileDefDD object) and security.
- Each binder contains documents (DKDocumentDefDD objects).
- Each document contains a profile (DKAttrProfileDefDD object) and security.
- Each profile contains fields (DKAttrFieldDefDD objects).
- Each field can contain keywords (DKAttrKeywordDefDD objects).

Listing entities and subentities

The following example lists the entities (in this example, rooms) in Domino.Doc and their subentities (in the example, cabinets, binders, and documents).

Java

```
...
// ----- Get a list of rooms
dkCollection rooms = dsDD.listEntities();
// ----- Iterate thru the rooms and their subEntities
dkIterator itRooms = rooms.createIterator();
itRooms.reset();
while( itRooms.more() ) {
    ... // Process the rooms and entities
}
```

The complete sample application from which this example was taken (TListSubEntitiesDD.java) is available in the samples directory.

C++

```
dkCollection* pColl = domDoc.listEntities();

long nbrEnts = pColl->cardinality();

dkIterator* itEnts = pColl-> createIterator();
while( itEnts->more() )
{ // For each returned dkEntityDef...
    DKRoomDefDD* pEnt = (DKRoomDefDD*)itEnts->next()->value();
    cout << "Room title: " << pEnt->getName() << endl;
    cout << "  Has SubEntities: " << pEnt->hasSubEntities() << endl;

    // print subEntities (Cabinets->Binders->Documents)
    printSubEnts(pEnt, domDoc, 1);

    delete pEnt;
}
delete itEnts;
delete pColl;
```

The complete application from which this example was taken (TListEntitiesDD.cpp) is available in the samples directory.

The following example lists document attributes and keywords:

Java

```
...
DKAttrProfileDefDD profile = aDocument.getProfile();
dkCollection fields = profile.getFields();

if((fields != null) &&( fields.cardinality() > 0 ))
{
    dkIterator itFields = fields.createIterator();
    while( itFields.more() )
    {
        DKAttrDefDD aField = (DKAttrDefDD)itFields.next();
        // ---- get the keywords
        dkCollection keywords = ((DKAttrFieldDefDD)aField).getKeywords();
        if( keywords != null )
        {
            if( keywords.cardinality() > 0 )
            {
                dkIterator itKeywords = keywords.createIterator();
                while( itKeywords.more() )
                {
                    DKAttrDefDD aKeyword = (DKAttrDefDD)itKeywords.next();
                    // ----- Process the keyword
                }
            }
        }
    }
}
...
```

The following example lists the subentities (cabinets, binders, and documents) associated with an entity (in this case a room).

C++

```
void printSubEnts( DKEntityDefDD* pEnt, DKDatastoreDD& domDoc, int indents )
{
    // indents: 1=Cabinets; 2=Binders; 3=Documents
    DKString indentation = "";

    for(int i = 0; i < indents; i++)
    {
        indentation += " ";
    }

    if( pEnt->hasSubEntities() )
    {
        dkCollection* pColl = pEnt->listSubEntities();
        long nbrEnts = pColl->cardinality();
        dkIterator* itEnts = pColl-> createIterator();
        while( itEnts->more() )
        {
            DKEntityDefDD* pEnt = (DKEntityDefDD*)itEnts->next()->value();
            cout<< indentation << "SubEntity title: " << pEnt->getName() << endl;
            printSubEnts(pEnt, domDoc, indents+1);
            delete pEnt;
        }
        delete itEnts;
        delete pColl;
    }
    return;
}
```

Listing cabinet attributes

Cabinets are the only items that contain any useful attributes. If you try to list entity attributes for rooms, nothing will appear in the collection. Therefore, when DKDatastoreDD lists searchable entities, it only lists cabinets.

The following example lists cabinets and their attributes.

Java

```
...
dkCollection cabinets = dsDD.listSearchableEntities();
dkIterator itCabinets = cabinets.createIterator();
while( itCabinets.more() )
{
    // ----- For each cabinet, list it's attributes.
    dkEntityDef aCabinet = (dkEntityDef)itCabinets.next();
    cabinetName = aCabinet.getName();
    // ----- List Document Profiles without sub-attributes
    System.out.println("\n" + Me + ": calling listAttrs for" + cabinetName );
    DKSequentialCollection coll=(DKSequentialCollection) aCabinet.listAttrs();
    ...
}
```

The complete sample application from which this example was taken (TListAttributes.java) is available in the samples directory.

Building queries in Domino.Doc

ENTITY= must be the first word in the query string if you want to limit the query to one cabinet. If the ENTITY parameter and its value are missing, then the entire library is searched. Also, the value must be enclosed in quotation marks ("). For example, "Diane Cabinet".

QUERY= is a required parameter.

In Domino.Doc a query string looks like this:

```
"ENTITY=<"cabinetTitle"> QUERY=<"lotusQueryString">"
```

Use the FTSearch function to query the Domino.Doc content server. The Domino.Doc content server must be fully text indexed for this function to work efficiently. To test for an index, use the IsFTIndexed property. To create an index, use the UpdateFTIndex function.

The FTSearch function searches all of the documents in a content server—to search documents within a particular view, use the FTSearch function in NotesView. To search documents within a particular document collection, use the FTSearch function in NotesDocumentCollection.

If you do not specify a sort option, documents are sorted by relevance. If you want to sort by date, you do not get relevance scores with the sorted results. If you pass the resulting DocumentCollection to a NotesNewsletter instance, results are sorted by either the document creation date or the relevance score, depending on which sort options you use.

Using query syntax

The syntax rules for a query are in the following list. Use parentheses to override precedence and to group operations.

Plain text

Use plain text to search for a word or phrase as-is. Enclose search keywords and symbols in apostrophes ('). Remember to use quotation marks (") whenever you are inside a LotusScript literal.

Wildcards

Use the question mark (?) to match any single character in any position within a word. Use the asterisk (*) to match zero to *n* (where *n* is any number) characters in any position in a word.

Logical operators

Use logical operators to expand or restrict your search. The operators and their precedents are:

1. ! (not)
2. & (and)
3. , (accrue)
4. | (or)

You can use either the keyword or symbol.

Proximity operators

Use proximity operators to search for words that are close to each other. These operators require word, sentence, and paragraph breaks in a full-text index. The operators are:

- near
- sentence
- paragraph

Field operator

Use the field operator to restrict your search to a specified field. The syntax is `FIELD field-name operator`, where *operator* is CONTAINS for text and rich text fields, and is one of the following symbols for number and date fields: =, >, >=, <, <=

Exactcase operator

Use the exactcase operator to restrict a search for the next expression to the specified case.

Termweight operator

Use the termweight *n* operator to adjust the relevance ranking of the expression that follows, where *n* is 0-100.

Working with relational databases

The Information Integrator for Content API classes support IBM DB2 Universal Database, and other relational databases using Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC) for C++.

Connecting to relational databases

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

Java

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

C++

```
try {      DKDatastoreDB2 dsDB2;
    dsDB2.connect("sample", userid, pw);
    . . .
    dsDB2.disconnect();
}
catch(DKException &exc) . . .
```

Use the database name when connecting.

Connection strings

When connecting to a relational database, you can specify a connection string and pass it as a parameter. If you specify multiple connection strings, separate them with a semi-colon (;). Connection strings can take the following forms:

Connection strings for DB2 UDB 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 UDB and ODBC:

CC2MIMEURL=(*URL*)

Specifies the cmbcc2mime.ini file as a uniform resource locator address. Use this form of the configuration string or CC2MIMEFILE, depending on the location of the file.

CC2MIMEFILE=(*filename*)

Specifies the cmbcc2mime.ini file by name.

DSNAME=(*content server name*)

Specifies the name of the content server. For federated queries and other federated functions, Information Integrator for Content sets this automatically.

AUTOCOMMIT=ON | OFF

Sets autocommit on or off. Default is off. When this content server is used for federated queries and other federated functions, autocommit is on by default.

Configurations strings for JDBC:

CC2MIMEURL=(*URL*)

Specifies the cmbcc2mime.ini file as a uniform resource locator address. Use this form of the configuration string or CC2MIMEFILE, depending on the location of the file.

Specifies the cmbcc2mime.ini file by name.

JDBCSEVERURL=(*URL*)

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

JDBCSEVERFILE=(*filename*)

Specifies the cmbjdbcsrvs.ini file that contains the list of JDBC servers as a filename.

JDBCDRIVER=(*JDBC driver*)

Specifies the JDBC driver that you want to use. This is automatically set when you use the system administration client client program.

DSNAME=(*content server name*)

Specifies the name of the content server. For federated queries and other federated functions, Information Integrator for Content sets this automatically.

AUTOCOMMIT=ON | OFF

Sets autocommit on or off. Default is off. When this content server is used for federated queries and other federated functions, autocommit is on by default.

Listing entities and entity attributes

After creating the content server for the relational database and connecting to it, you can list the entity and entity attributes. The following example shows how to retrieve list and step through it:

Java

```
// ----- After creating a datastore and connecting, get index classes
pCol = (DKSequentialCollection)dsDB2.listDataSources();
pIter = pCol.createIterator();
while (pIter.more() == true)
{
    i++;
    pSV = (DKServerDefDB2)pIter.next();
    strServerName = pSV.getName();
    .... // Use the server name as appropriate
}
// ----- Connect to datastore
dsDB2.connect(db, userid, pw, "");
if (!schema.equals(""))
{
    dsDefDB2 = (DKDatastoreDefDB2)dsDB2.datastoreDef();
    dsDefDB2.setSchemaName(schema);
    schema = dsDefDB2.getSchemaName();
    System.out.println(" New Schema Name = [" + schema + "]");
}
// ----- List the tables
pCol = (DKSequentialCollection) dsDB2.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
    i++;
    tableDef = (DKTableDefDB2)pIter.next();
    strTable = tableDef.getName();
    // ----- List attributes (columns for the table)
    pCol2 = (DKSequentialCollection) dsDB2.listEntityAttrs(strTable);
    pIter2 = pCol2.createIterator();
    j = 0;
    while (pIter2.more() == true)
    {
        j++;
        colDef = (DKColumnDefDB2)pIter2.next();
        .... // Process the information as appropriate
    }
}
// ----- Commit and disconnect
dsDB2.commit();
dsDB2.disconnect();
dsDB2.destroy();
```

Refer to `TListCatalogDB2.java`, `TListCatalogJDBC.java`, and `TListCatalogDJ.java` in the samples directory for complete examples.

C++

```
try {
    DKDatastoreDB2 dsDB2;
    DKString schema;
    DKSequentialCollection *pCol2 = 0;
    dkIterator *pIter = 0;
    dkIterator *pIter2 = 0;
    DKTableDefDB2 *pEnt = 0;
    DKString strServerName;
    DKString strTable;
    DKColumnDefDB2 *pAttr = 0;
    DKDatastoreDefDB2 *dsDefDB2 = 0;
    DKAny a;
    DKAny *pA = 0;
    long i = 0;
    long j = 0;

    // ----- Connect to datastore and set value for schema name
    . . .
    // ----- Create a datastore definition and set the schema name
    dsDefDB2 = (DKDatastoreDefDB2 *) dsDB2.datastoreDef();
    if (schema != "")
    {
        dsDefDB2->setSchemaName(schema);
    }

    // ----- Get a list of the entities (tables)
    pCol = (DKSequentialCollection*)((dkCollection*)dsDB2.listEntities());
    pIter = pCol->createIterator();
    i = 0;
    // ----- List the attributes (columns) for each entity (table)
    while (pIter->more() == TRUE)
    {
        i++;
        pEnt = (DKTableDefDB2*)((void*)(*pIter->next()));
        strTable = pEnt->getName();
        cout << "table name [" << i << "] - " << strTable << endl;
        cout << " list columns for " << strTable << " table" << endl;
        pCol2 =
        (DKSequentialCollection*)((dkCollection*)dsDB2.listEntityAttrs(strTable));
        pIter2 = pCol2->createIterator();
        j = 0;
        while (pIter2->more() == TRUE)
        {
            j++;
            pA = pIter2->next();
            pAttr = (DKColumnDefDB2*) pA->value();
            cout << "    Attr name [" << j << "] - " << pAttr->getName() << endl;
            cout << "        datastoreName " << pAttr->datastoreName() << endl;
            cout << "        datastoreType " << pAttr->datastoreType() << endl;
            cout << "        attributeOf " << pAttr->getEntityName() << endl;
            cout << "        type " << pAttr->getType() << endl;
            cout << "        size " << pAttr->getSize() << endl;
            cout << "        id " << pAttr->getId() << endl;
            cout << "        nullable " << pAttr->isNullable() << endl;
            cout << "        precision " << pAttr->getPrecision() << endl;
            cout << "        scale " << pAttr->getScale() << endl;
            cout << "        string type " << pAttr->getStringType() << endl;
            cout << "        primary key " << pAttr->isPrimaryKey() << endl;
            cout << "        foreign key " << pAttr->isForeignKey() << endl;
            delete pAttr;
        }
    }
    // continued...
```

C++ (continued)

```
        delete pIter2;
        delete pCol2;
        delete pEnt;
    }
    delete pIter;
    delete pCol;
    dsDB2.disconnect();
}
catch(DKException &exc)
{
    . . .
}
```

Refer to TListCatalogDB2.cpp, TListCatalogODBC.cpp, and TListCatalogDJ.cpp in the samples, odbc, and dj directories for complete examples.

Running a query

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

Java

```
// ----- After creating a datastore and connecting, build the
//          query and parameters and execute it
sDB2 = new DKDatastoreDB2();
dkResultSetCursor pCur = null;
DKNVPair parms[] = new DKNVPair[2];
String strMax = "5";
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS, strMax);
parms[1] = new DKNVPair(DK_CM_PARM_END, null);
// ----- Connect to datastore
dsDB2.connect(database, userid, pw, "");
// --- Create the query string
String cmd = "";
cmd = "SELECT * FROM EMPLOYEE";

DKDDO p = null;
DKDDO pDDO = null;
dkXDO pXDO = null;
DKPidXDO pidXDO = null;
int i = 0;
int numDataItems = 0;
short k = 0;
short j = 0;
String strDataName;
dkCollection pCol = null;
dkIterator pIter = null;
Object a = null;
dkDataObjectBase pDO = null;
int cnt = 0;

// continued...
```

Java (continued)

```
// ----- Execute the query
pCur = dsDB2.execute(cmd,DK_CM_SQL_QL_TYPE,parms);
if (pCur == null)
{
    // Handle if the cursor is null
}
while (pCur.isValid())
{
    p = pCur.fetchNext();
    if (p != null)
    {
        cnt++;
        i = pCur.getPosition();
        // Get item information
        numDataItems = p.dataCount();
        DKPid pid = p.getPidObject();
        System.out.println("pid string " + pid.pidString());
        System.out.println("Number of Data Items " + numDataItems);
        for (j = 1; j <= numDataItems; j++)
        {
            a = p.getData(j);
            strDataName = p.getDataName(j);
            // Handle the attributes ;
            if (a instanceof String)
            {
                System.out.println("    Attribute Value " + a);
            }
            ..... // Handle for various types )
            else if (a instanceof dkDataObjectBase)
            {
                pDO = (dkDataObjectBase)a;
                if (pDO.protocol() == DK_PDDO)
                {
                    System.out.println("    DKDDO object ");
                    pid = ((DKDDO)pDO).getPidObject();
                }
                else if (pDO.protocol() == DK_XDO)
                {
                    // dkXDO object
                    pXDO = (dkXDO)pDO;
                    pidXDO = pXDO.getPidObject();
                }
            }
            ..... // Handle for various types
        }
    }
}
// Delete the cursor when you're done, commit and disconnect
pCur.destroy(); // Finished with the cursor
dsDB2.commit();
dsDB2.disconnect();
dsDB2.destroy();
```

Refer to TExecuteDB2.java, TExecuteJDBC.java, and TExecuteDJ.java in the samples directory for complete examples.

C++

```
try {
    DKDatastoreDB2 dsDB2;
    dkResultSetCursor* pCur = 0;
    DKNVPair par[2];
    DKAny anyValue;
    DKString strMax = "5";
    anyValue = strMax;
    par[0].set(DK_CM_PARM_MAX_RESULTS, anyValue);
    par[1].setName(DK_CM_PARM_END);
    // ---- Create a datastore and connect
    . . .
    // ---- Create a query string containing the select
    DKString qstrng = "SELECT * FROM EMPLOYEE";
    // ---- Execute the query
    pCur = dsDB2.execute(qstrng, DK_CM_SQL_QL_TYPE, par);
    // ---- Declarations
    DKDDO *p = 0;
    dkDataObjectBase *pDOBase = 0;
    DKDDO *pDDO = 0;
    dkXDO *pXDO = 0;
    DKAny a;
    ushort j = 0;
    ushort k = 0;
    ushort val = 0;
    ushort cnt = 1;
    DKString strData = "";
    DKString strDataName = "";
    dkCollection* pdCol = 0;
    dkIterator* pdIter = 0;
    ushort numDataItems = 0;
    DKString strPid;
    DKPid* pid = 0;
    short sVal = 0;
    long lVal = 0;
    while (pCur->isValid())
    {
        p = pCur->fetchNext();
        if (p != 0)
        {
            cout << "======" << "Item " << cnt << " <======" << endl;
            numDataItems = p->dataCount();
            pid = (DKPid*)p->getPidObject();
            strPid = pid->pidString();
            cout << "pid string " << strPid << endl;
            k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
            if (k > 0)
            {
                a = p->getProperty(k);
                val = a;
                switch (val)
                {
                    case DK_CM_DOCUMENT :
                    {
                        cout << "Item is document " << endl;
                        break;
                    }
                }
            }
        }
        cout << "Number of Data Items " << numDataItems << endl;
    }
}
```

C++ (continued)

```
for (j = 1; j <= numDataItems; j++)
{
    a = p->getData(j);
    strDataName = p->getDataName(j);
    switch (a.typeCode())
    {
        case DKAny::tc_string :
        {
            strData = a;
            cout << "attribute name : " << strDataName << " value : "
<< strData << endl;
            break;
        }
        // ---- Handle each type in a similar fashion
        . . .
    }
}
// ----- Delete the cursor and disconnect
if (pCur != 0)
    delete pCur;
dsDB2.disconnect();
}
catch(DKException &exc)
    . . .
```

Refer to TExecuteDB2.cpp, TExecuteODBC.cpp, and TExecuteDJ.cpp in the sample directories for complete examples.

Creating custom content server connectors

You can create your own server definitions for custom content servers (that are not currently included in Information Integrator for Content). If you integrate a custom server into Information Integrator for Content, then you must provide your own Java or C++ classes to support the definition.

Developing custom content server connectors

The object-oriented API framework is designed with the following objectives:

- Additional data storage systems, or content servers, can be added into the framework.
- Ability to map to any complex content server data type.
- A common object model for all content server data access.
- A flexible mechanism to use a combination of different types of search engines, such as DB2 Text Information Extender, image search (QBIC), and so forth.
- Client/server implementation for Java application users.

For information on specific object-oriented APIs see the Application Programming Reference.

If you are integrating a custom content server into Information Integrator for Content you must:

- Import the `com.ibm.mm.sdk.common` package.
- **Java only:** Link to the `cmbcm81.jar` (Java) file in order to access the common framework.

- **C++ only:** Link to the cmbcm817.dll, non-debugged version, and cmbcm8167.dll, debugged version, files in order to access the common framework.

Information Integrator for Content database infrastructure

The dkDatastore classes serve as the primary interface between Information Integrator for Content and the content servers. Each content server has a separate class that implements the dkDatastore class to provide implementation information for a specific content server. Each content server type is represented by a class called DKDatastorexx, where xx identifies the name or type of the specific content server. Table 5 on page 23 lists the current content servers provided in Information Integrator for Content.

You must specify the DKDatastore class you create for your content server in the Information Integrator for Content system administration client when you create your server definition.

Common classes in Information Integrator for Content

dkDDO

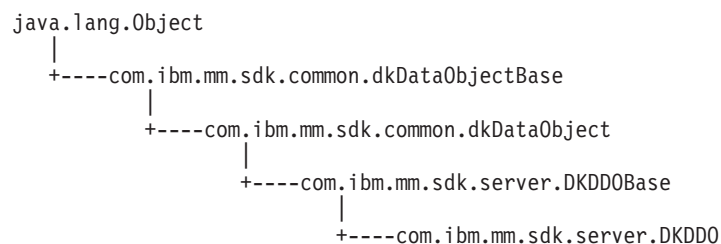
The dkDDO class provides a representation and a protocol to define and access an object's data, independent of the object's type. The DDO protocol is implemented as a set of functions to define, add, and access each data item of an object. You can use this protocol to dynamically create an object and 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 DB2 Content Manager content server, or a document in OnDemand.

Java

This diagram represents the hierarchy for the dkDDO class:



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 dkXDODBase by defining independent content server access, create, retrieve, update, and delete functions. These functions enable an application to store and retrieve the object's data to and from a content server without the existence of an associated DDO class object or stand-alone XDO.

You must set the PID for a stand-alone XDO to locate its position in the content server. If you are using the XDO with a DDO, the PID is set automatically. For example you can map which class to an item for the DB2 Content Manager content servers, and mapped to notes for the OnDemand content servers.

Java

Here is the class hierarchy for the dkXDO class:

```

java.lang.Object
|
+----com.ibm.mm.sdk.server.dkDataObjectBase
|
+----com.ibm.mm.sdk.server.dkXDObase
|
+----com.ibm.mm.sdk.server.dkXDO

```

dkCollection

The dkCollection class is a collection of objects. dkCollection cannot evaluate a query. A collection might have a name (the default name is an empty string). For example, DKParts is a subclass of DKSequentialCollection, which is in turn a subclass of dkCollection.

DKResults

DKResults is a subclass of dkQueryableCollection, therefore it supports sorting and bi-directional iterators, and it is queryable. The element members of a DKResults class are objects, instances of the dkDDO class that represent query results. The iterator created by this class is dkSequentialIterator.

Java

Here is the class hierarchy for the DKResults class:

```

java.lang.Object
|
+----com.ibm.mm.sdk.server.DKSequentialCollection
|
+----com.ibm.mm.sdk.server.dkQueryableCollection
|
+----com.ibm.mm.sdk.server.DKResults

```

dkQuery

dkQuery is an interface for a query object associated with a specific content server. Objects that implement this interface are created by content server classes. The result of a query is usually a DKResults object. Examples of a concrete implementation of the dkQuery interface are DKParametricQuery, DKTextQuery and DKImageQuery, which are created by their associated content servers.

dkCQExpr

The dkCQExpr class represents a compound or combined query expression. It can contain a dkQExpr query expressions tree, which can contain a combination of parametric, text, and image query. If you want each content server to allow a federated search, the content server must be able to process this dkCQExpr object.

dkSchemaMapping

dkSchemaMapping is the an interface that defines an associative mapping

between a federated entity and a native entity in content server. The content server must understand this mapping class to unmap and remap federated entities and attributes to native entities and attributes for a query and return results.

dkDatastore and related classes

You must implement one concrete class for each of the following classes or interfaces for your content server. For example in an OnDemand server, the concrete class that implements the dkDatastore interface is DKDatastoreOD.

dkDatastore

dkDatastore represents and manages a connection to the content server, its transactions and commands. It supports the evaluate function, so it can be considered a subclass of the query manager.

The main methods in the dkDatastore interface are:

connect()

Connects to the content server.

disconnect()

Disconnects from the content server.

evaluate(), execute(), executeWithCallback()

Queries the content server.

commit(), rollback()

Performs transactions in the content server.

Restriction: Some content servers do not support these functions.

registerServices(), unregisterServices()

Registers search engines.

changePassword(userid, oldPasswd, newPasswd)

Changes the login password for the current logon user ID from the content server.

listDataSources()

Returns a collection of content server user ID objects to use for logon. You do not need to be connected to the content server to use this function.

listDataSourceNames()

Returns an array of content server names.

getExtension(String)

Gets the dkExtension object from the content server. If the given extension does not already exist but is supported by the content server, a newly created object is returned, otherwise, a null value is returned.

addExtension(String, dkExtension)

Adds a new extension object (XDO) to this content server.

createDDO(String,int)

Creates a data object based on the given object type and flag. Create DDO returns a new DKDDO object with all the properties and attributes set. The calling program must provide the attribute values for this data object.

The data object manipulation methods in the dkDatastore interface are:

addObject(dkDataObject)

Adds a new document or folder to the content server.

retrieveObject(dkDataObject)

Retrieves a document or folder from the content server.

deleteObject(dkDataObject)

Deletes a document or folder from the content server.

updateObject(dkDataObject)

Updates a document or folder in the content server.

moveObject(dkDataObject, String)

Moves a folder or document from one entity to another.

The schema mapping related methods in the dkDatastore interface are:

registerMapping(DKNVPair)

Registers the mapping information to this content server.

unRegisterMapping(String)

Removes the mapping information from this content server.

listMappingNames()

Returns an array of mapping names from this content server.

getMapping(String)

Returns a dkSchemaMapping object.

dkDatastoreDef

The dkDatastoreDef interface defines functions to access content server information and to create, list, and delete its entities. It maintains a collection of dkEntityDef objects.

Table 28 contains examples of concrete classes for the dkDatastoreDef interface.

Table 28. Concrete classes for dkDatastoreDef

Server type	Class name
DB2 Content Manager	DKDatastoreDefICM
OnDemand	DKDatastoreDefOD
DB2 Content Manager for AS/400	DKDatastoreDefV4
ImagePlus for OS/390	DKDatastoreDefIP
Domino.Doc	DKDatastoreDefDD
Extended Search	DKDatastoreDefDES
IBM DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC
Earlier DB2 Content Manager	DKDatastoreDefDL

The main methods in the dkDatastoreDef interface are:

listEntities()

Lists entities.

listEntityAttrs()

Lists entity attributes.

addEntity()

Adds an entity.

getEntity(name)

Gets an entity.

Each concrete class can also have its own content server-specific functions with names that are familiar to that content server. For example, the DKDatastoreDefDL class contains these specific functions:

- listIndexClassNames()
- listIndexClasses()

The DKDatastoreDefOD class contains these specific functions:

- listAppGrps()
- listAppGrpNames()

dkEntityDef

The dkEntityDef class defines functions to:

- Access entity information.
- Create and delete attributes.
- Create and delete the entity.

In the dkEntityDef class, all functions that are related to subentities generate a DKUsageError indicating that the default content server does not support subentities. However, if the content server does support this kind of multiple level entity, as does Domino.Doc, for example, the subclass for this content server must implement the proper functions to overwrite the exceptions.

Table 29 contains examples of concrete classes for the dkEntityDef class.

Table 29. Concrete classes for dkEntityDef

Server type	Class name
DB2 Content Manager	DKItemTypeDefDL
OnDemand	DKAppGrpDefOD
DB2 Content Manager for AS/400	DKIndexClassDefV4
ImagePlus for OS/390	DKEntityDefIP
Domino.Doc	DKCabinetDefDD
Extended Search	DKDatabaseDefDES
DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC
Earlier DB2 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 30 contains examples of concrete classes for the dkAttrDef class.

Table 30. Concrete classes for dkAttrDef

Server type	Class name
DB2 Content Manager	DKAttrDefDICM
OnDemand	DKFieldDefOD
DB2 Content Manager for AS/400	DKAttrDefV4
ImagePlus for OS/390	DKAttrDefIP
Domino.Doc	DKAttrDefDD
Extended Search	DKFieldDefDES
DB2 Universal Database	DKColumnDefDB2
JDBC	DKColumnDefJDBC
ODBC	DKColumnDefODBC
Earlier DB2 Content Manager	DKAttrDefDL

The main methods in the dkAttrDef class are:

listAttrs()
Lists the attributes.

getAttr(*String attrName*)
Gets a specified attribute.

getName()
Gets the name of the attribute.

getDescription()

Gets the description of the attribute.

add() Adds the entity to the content server.

dkServerDef

The dkServerDef class provides the server definition information for each content server. Table 31 contains examples of concrete classes for the dkServerDef class.

Table 31. Concrete classes for dkServerDef

Server type	Class name
DB2 Content Manager	DKServerDefICM
OnDemand	DKServerDefOD
DB2 Content Manager for AS/400	DKServerDefV4
Domino.Doc	DKServerDefDD
Extended Search	DKServerDefDES
DB2 Universal Database	DKServerDefDB2
JDBC	DKServerDefJDBC
ODBC	DKServerDefODBC
earlier DB2 Content Manager	DKServerDefDL

The main functions in the dkServerDef class are:

setDatastore(dkDatastore ds)

Sets the reference to the content server object.

getDatastore()

Gets the reference to the content server object.

getName()

Gets the name of the content server.

setName(String name)

Sets the name of the content server.

datastoreType()

Gets the content server type.

dkResultSetCursor

dkResultSetCursor is a content server cursor in the query result set that you can use to manage a virtual collection of DDO objects. The collection is a query result set. Each element of the collection is not created until the content server retrieves the element.

The main functions in the dkResultSetCursor class are:

isScrollable()

Returns TRUE if the cursor can be scrolled forward and backward.

isUpdatable()

Returns TRUE if the cursor can be updated.

isValid()

Returns TRUE if the cursor is valid.

isBegin()

Returns TRUE if the cursor is positioned at the beginning of the result set.

isEnd()
Returns TRUE if the cursor is positioned at the end of the result set.

isInBetween()
Returns TRUE if cursor is positioned between data elements in the result set.

getPosition()
Gets the current position of the cursor.

setPosition(int position, Object value)
Sets the cursor to the specified position.

setToNext()
Sets the cursor to point to the next element in the result set.

fetchObject()
Retrieves the current element from the result set and returns it as a DDO.

fetchNext()
Retrieves the next element from the result set and returns it as a DDO.

findObject(int position, String predicate)
Finds the data object that satisfies the specified predicate, moved the cursor to that position, and then retrieves the object.

addObject(DKDDO ddo)
Adds a new element of the same type, represented by the specific DDO, to the content server.

deleteObject()
Deletes the current element from the content server.

updateObject(DKDDO ddo)
Updates the current element at the current position in the content server, using the specified DDO.

newObject()
Creates an element of the same type and returns it as a DDO.

open() Opens the cursor, and if necessary, runs the query to create the result set.

close() Closes the cursor and the result set.

isOpen()
Returns TRUE if the cursor is open.

destroy()
Deletes the cursor; this allows for cleanup before the cursor is collected as garbage.

datastoreName()
Gets the name of the content server name to which the cursor belongs.

datastoreType()
Gets the content server type to which the cursor belongs.

handle(int type)
Gets the result set handle that is associated with the result set cursor, by type.

Requirement: In order to use the `addObject`, `deleteObject` and `updateObject` functions, you must set the content server option `DK_DL_OPT_ACCESS_MODE` to `DK_READWRITE`.

dkBlob

`dkBlob` is an abstract class that declares a common public interface for basic binary large object (BLOB) data types. The concrete classes derived from the `dkBlob` class share this common public interface which allows polymorphic processing of collections of BLOBs originating from heterogeneous content servers. There is also a `dkClob` and a `dkDBClob` class which can have concrete classes.

Table 32 contains examples of concrete classes for the `dkBlob` class.

Table 32. Concrete classes for `dkBlob`

Server type	Class name
DB2 Content Manager	DKLobICM
OnDemand	DKBlobOD
DB2 Content Manager for AS/400	DKBlobV4
ImagePlus for OS/390	DKBlobIP
Domino.Doc	DKBlobDD
Extended Search	DKBlobDES
DB2 Universal Database	DBBlobDB2, DKBlobDB2
JDBC	DKBlobJDBC, DKBlobJDBC
ODBC	DKBlobODBC, DKBlobODBC
Earlier DB2 Content Manager	DKBlobDL

The main methods in the `dkBlob` class are:

getContent()

Returns a byte array containing the BLOB data of the object.

getContentToClientFile(String afileName, int fileOption)

Copies the BLOB data from the object to the specified file.

setContent(byte[] aByteArr)

Sets the LOB data for the object with the contents of the byte array.

setContentFromClientFile(String afileName)

Replaces the LOB data of the object with the contents of the file *afileName*.

add(String afileName)

Adds the content of the specified file to the content server.

retrieve(String afileName)

Retrieves the content of the content server into the specified file.

update(String afileName)

Updates the object and the content server with the content of the specified file

del(boolean flush)

Deletes the object's data from the content server, if *flush* is `TRUE`; otherwise the current content is preserved.

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

Concatenates this object with another `dkBlob` object or byte array.

length()

Returns the length of the LOB content of the object.

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

Starting the search at offset *startPos*, returns the byte offset of the first occurrence of the search argument within this object.

substring(int startPos, int length)

Returns a string object that contains a substring of the LOB data of this object.

remove(int startPos, int aLength)

Starting at *startPos* for *aLength* bytes, deletes a portion of the LOB data of this object.

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

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

open(String afileName)

Unloads the object contents to the file *afileName* and then runs a default file handler.

setClassOpenHandler(String aHandler, boolean newSynchronousFlag)

Identifies, by executable program name, the file handler for an entire class. This 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 33 contains examples of concrete classes for the dkClob class.

Table 33. Concrete classes for dkClob

Server type	Class name
DB2 Universal Database	DKClobDB2
ODBC	DKClobODBC

The main functions in the dkClob class are:

open() Open() is a member inherited from dkXDOBase. Open() will be implemented or overridden by concrete subclasses of dkClob.

dkXDO Members: `dkXDO& add()`, `dkXDO retrieve()`, `dkXDO update()`, `dkXDO del()`

Inherited as protected members from `dkXDO`. Where necessary, these protected members will be implemented or overridden by concrete subclasses of `dkClob`.

The following list contains members defined by `dkClob`:

`add(String afileName)`

Adds the content of the specified file to the content server.

`retrieve(String afileName)`

Retrieves the content of the content server into the specified file.

`update(String afileName)`

Updates the object and the content server with the content of the specified file.

`del(DKBoolean flush)`

Deletes the object's data from the content server, if *flush* is `TRUE`; otherwise the current content is preserved.

`getContentToClientFile(String afileName, int fileOption)`

Copies the CLOB data from the object to the specified file.

`setContentFromClientFile(String afileName)`

Replaces the LOB data of the object with the contents of the file *afileName*.

`indexOf(String& aString, long startPos=1, indexOf(dkClob& adkClob, long startPos=1)`

Starting the search at offset *startPos*, returns the byte offset of the first occurrence of the search argument within this object.

`substring(long startPos, long length)`

Returns a string object that contains a substring of the LOB data of this object.

`remove(long startPos, long aLength)`

Starting at *startPos* for *aLength* bytes, deletes a portion of the LOB data of this object.

`insert(DKString aString, long startPos), insert(dkClob& adkClob, long startPos)`

Inserts the argument data following the *startPos* position in the CLOB data of the object.

`open(String afileName)`

Unloads the object contents to the file *afileName* and then runs a default file handler.

`setInstanceOpenHandler(String ahandler, DKBoolean newSynchronousFlag)`

Identifies, by executable program name, the file handler and indicates whether to run it synchronously or asynchronously for this object.

`setClassOpenHandler(String ahandler, DKBoolean newSynchronousFlag)`

Identifies, by executable program name, the file handler for an entire class. This 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 DKBlobxx class, where the dkBlob object is the representation of the binary annotation data and the DKParts collection.

dkDatastoreExt

The dkDatastoreExt class defines the standard content server extension classes.

Table 34 contains examples of concrete classes for the dkDatastoreExt class.

Table 34. Concrete classes for dkDatastoreExt

Server type	Class name
DB2 Content Manager	DKDatastoreExtICM
OnDemand	DKDatastoreExtOD
DB2 Content Manager for AS/400	DKDatastoreExtV4
ImagePlus for OS/390	DKDatastoreExtIP
Domino.Doc	DKDatastoreExtDD
Extended Search	DKDatastoreExtDES
DB2 Universal Database	DKDatastoreExtDB2
JDBC	DKDatastoreExtJDBC
Earlier DB2 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 35 contains examples of concrete classes for the DKPidXDO class.

Table 35. Concrete classes for DKPidXDO

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

dkUserManagement

The dkUserManagement class represents and processes all of the content server's user management functions.

Table 36 contains examples of concrete classes for the dkUserManagement class.

Table 36. Concrete classes for dkUserManagement

Server type	Class name
DB2 Content Manager	DKUserMgmtICM
DB2 Content Manager for AS/400	DKUserMgmtV4

Table 36. Concrete classes for dkUserManagement (continued)

Server type	Class name
ImagePlus for OS/390	DKUserMgmtIP
Earlier DB2 Content Manager	DKUserMgmtDL

DKConstant

All common constants are defined in the DKConstant class. Each content server has its own DKConstantxx class for defining constants specific to that content server.

Recommendation: All content servers use the common messages whenever possible.

DKMessageId

All common message IDs are defined in this class. Each content server has its own DKMessageIdxx class for defining its own message IDs.

Recommendation: All content servers should use the common messages whenever possible.

These property files contain common warning and error messages:

For Java:

- DKMessage_en.properties
- DKMessage_es.properties

For C++:

- DKMessage_en_US.properties
- DKMessage_es_ES.properties

Each content server has its own DKMessagexx_yy_zz.properties files for its warning and error messages.

Using the FeServerDefBase class (Java only)

The FeServerDefBase class is the abstract class that you must extend in order to create a custom server definition. The Java class that extends this base class must have a constructor that accepts the following parameters and passes them to the super class:

String connectString

The connect string for the server.

String[] serverList

The list of defined servers.

String[] associatedServerList

The list of servers associated with this server (null if none).

String[] serverTypes

The list of defined server type IDs.

String[] serverTypeDescriptions

The list of descriptions for defined server types.

When you create the Java class that extends the FeServerDefBase class you must determine how to handle the data for the new server dialog. You can use the same class or a separate model class. If the custom content server requires more than

fields for the connect string, you must use the Information Integrator for Content database and Java APIs as a model in order for additional functions to perform properly.

When the content servers are selected in the Information Integrator for Content Administration program, the New menu will contain the list of server types stored in the FASERVERTYPES table in the Information Integrator for Content 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 Information Integrator for Content 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 9. Building Information Integrator for Content workflow applications

Using the Information Integrator for Content classes and APIs, you can create or extend your own applications to use the Information Integrator for Content workflow support. Typically, you perform a federated search, and start the workflow with the search result (a content item or a folder of multiple content items). You use the APIs to access a worklist and then to display the worklist contents. As each activity completes, the workflow moves to the next activity in the workflow.

Connecting to workflow services

To use DB2 Information Integrator for Content workflow in your applications, start by creating an instance of `DKWorkFlowServicesFed`, then connect to it. The following example starts workflow services:

Java

```
// ----- Create the strings for the name of the
//service, user ID and Password
String wfsrv = "icmnlsdb";
String userid = "icmadmin";
String pw = "password";
// ----- Create a federated datastore
DKDatastoreFed dsFed = new DKDatastoreFed();
dsFed.connect(wfsrv, userid, pw,"");
//----- Create the workflow service
DKWorkFlowServiceFed svWF =new DKWorkFlowServiceFed ();
// ----- Set the datastore in the workflow service
svWF.setDatastore(dsFed);
// ----- Connect to the service
svWF.connect (wfsrv, userid, pw,"");
```

C++

```
// ----- Create the strings for the name of the service, user ID
// ----- and Password
DKString wfsrv = "icmnlsdb";
DKString userid = "icmadmin";
DKString pw = "password";
// ----- Create a federated datastore
DKDatastoreFed* dsFed = new DKDatastoreFed();
dsFed->connect(wfsrv, userid, pw,"");
//----- Create the workflow service
DKWorkFlowServiceFed* svWF =new DKWorkFlowServiceFed ();
// ----- Set the datastore in the workflow service
svWF->setDatastore(dsFed);
// ----- Connect to the service
svWF->connect (wfsrv, userid, pw,"");
```

When you are finished using the workflow service, you must disconnect by calling the `disconnect()` and the `delete()` functions.

Java

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

C++

```
svWF->disconnect();  
dsFed->disconnect();  
delete svWF;  
delete dsFed;
```

Starting a workflow

After you create the workflow, you must start it. To start a workflow complete the following steps:

1. Create a DKWorkFlowFed object and set the workflow name.
2. Create a workflow instance using a valid workflow template, which is a workflow definition defined in the DB2 Information Integrator for Content workflow builder.
3. Set the PID and priority in the container.
4. Start the workflow.

The following example uses these steps to start a workflow:

Java

```
// ----- Create the DKWorkFlowFed object and set the name  
DKWorkFlowFed WF = new DKWorkFlowFed(svWF);  
WF.setName("wf1");  
// ----- Create an instance of a workflow with the workflow template name  
WF.add("WD1");  
// ----- Refresh the workflow object  
WF.retrieve();  
// ----- Construct the container object for the workflow  
DKWorkFlowContainerFed con = WF.inContainer();  
// ----- Retrieve the container data  
con.retrieve();  
// ----- Add a PID string referring to an Extended Search document  
con.setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");  
con.setPriority(100);  
// ----- Update the container  
con.update();  
// ----- Start the workflow  
WF.start(con);
```

C++

```
// - Create the DKWorkflowFed object and set the name
DKWorkflowFed* WF = new DKWorkflowFed(svWF);
WF->setName("wfl");
//Create an instance of a workflow with the workflow template name
WF->add("WD1");
// ----- Refresh the workflow object
WF->retrieve();
// ----- Construct the container object for the workflow
DKWorkflowContainerFed* con = WF.inContainer();
// ----- Retrieve the container data
con->retrieve();
// Add a PID string referring to a content item from Extended Search
con->setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
// ----- Assign a priority of 100
con->setPriority(100);
// ----- Update the container
con->update();
// ----- Start the workflow
WF->start(con);
. . .
// When you are done, clean up by deleting the container and workflow
delete con;
delete WF;
```

Terminating a workflow

You can terminate a workflow by calling the `terminate()` or `del()` function as shown in the following example:

Java

```
//-----Retrieve the status of the workflow named WF
WF.retrieve();
int state =WF.state();
//-----Check the status and either terminate or delete
if (state ==DKConstantFed.DK_FED_FMC_PS_RUNNING ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDED ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDING)
{
    WF.terminate();
}
if (state ==DKConstantFed.DK_FED_FMC_PS_READY ||
    state ==DKConstantFed.DK_FED_FMC_PS_FINISHED||
    state ==DKConstantFed.DK_FED_FMC_PS_TERMINATED)
{
    WF.del();
}
```


C++

```
/--Construct a DKWorkflowFed instance
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
//-----Retrieve the status of the workflow named WF
WF->retrieve();
int state = WF->state();
/--Check the status and either terminate or delete
if (state == DK_FED_FMC_PS_RUNNING ||
state == DK_FED_FMC_PS_SUSPENDED ||
state == DK_FED_FMC_PS_SUSPENDING)
{
WF->terminate();
}
if (state == DK_FED_FMC_PS_READY ||
state == DK_FED_FMC_PS_FINISHED ||
state == DK_FED_FMC_PS_TERMINATED)
{
WF->del();
}
delete WF;
```

Listing all the workflows

You can list all the workflows in a workflow service by using the `listWorkFlows()` function. The following example lists the name and description of all the workflows in a workflow service referenced by the `DKWorkflowServiceFed` object `svWF`.

Java

```
// ----- Call the listWorkFlows method
DKSequentialCollection collWF = (DKSequentialCollection)svWF.listWorkFlows();
DKWorkflowFed WF = null;
if (collWF != null)
{
    dkIterator iterWF = collWF.createIterator();
    while (iterWF.more() == true)
    {
        WF = (DKWorkflowFed)iterWF.next();
        WF.retrieve();
        System.out.println("name = " + WF.getName() + " description = "
                           + WF.getDescription());
    }
    iterWF = null;
}
```

C++

```
// ----- Call the listWorkFlows function
DKSequentialCollection *collWF =
    (DKSequentialCollection*)svWF.listWorkFlows();
DKWorkflowFed *WF = NULL;
if (collWF != NULL)
{
    dkIterator *iterWF = collWF->createIterator();
    while (iterWF->more())
    {
        WF = (DKWorkflowFed*)(void*)(*iterWF->next());
        WF->retrieve();
        cout << "name = " + WF->getName()
        << " description = " << WF->getDescription() << endl;
        delete WF;
    }
    delete iterWF;
}
delete collWF;
```

Suspending a workflow

You can suspend a running workflow with either a specific time or indefinitely. The following example shows how to suspend a workflow until a certain time. If you provide a null `DKTimestamp`, then Information Integrator for Content suspends the workflow indefinitely.

Java

```
// ----- Construct a DKWorkflowFed object
DKWorkflowFed WF = new DKWorkflowFed(svWF, "Test");
WF.retrieve();
// ----- Call the suspend method if the workflow is in the running state
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_RUNNING)
{
    // ----- Suspended until 2000-07-27-16.30.00.000000
    // ----- The timestamp uses the base year 1900; months are
    // ----- numbered 0 to 11
    DKTimestamp suspension = new DKTimestamp(100, 6, 27, 16, 30, 0, 0);
    WF.suspend(suspension);
}
```

C++

```
// ----- Construct a DKWorkflowFed instance
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
WF->retrieve();
// ----- Call the suspend function if the workflow is in
the running state
if (WF->state() == DK_FED_FMC_PS_RUNNING)
{
    // ----- Suspended until 2000-07-27-16.30.00.000000
    DKTimestamp* suspension = new DKTimestamp(2000, 7,
    27, 16, 30, 0, 0);
    WF->suspend(suspension);
    delete suspension;
}
delete WF;
```

Resuming a workflow

You can resume a suspended workflow by calling the `resume()` function. The following example resumes a suspended workflow.

Java

```
// ----- Construct a DKWorkflowFed object
DKWorkflowFed WF = new DKWorkflowFed(svWF, "Test");
WF.retrieve();
// ---- Call resume() if the workflow is in the suspended state
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_SUSPENDED)
{
    WF.resume();
}
```

C++

```
// ----- Construct a DKWorkflowFed instance
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
WF->retrieve();
// ----- Check whether the workflow is suspended and call resume
if (WF->state() == DK_FED_FMC_PS_SUSPENDED)
{
    WF->resume();
}
delete WF;
```

Listing all the worklists

You can list all the worklists in a workflow service by calling the `listWorkLists()` function on the workflow service. The following example lists the name and description of all the worklists in a workflow service referenced by the `DKWorkflowServiceFed` instance `svWF`.

Java

```
// ----- Call the listWorkLists method
DKSequentialCollection collWL = (DKSequentialCollection)svWF.listWorkLists();
DKWorkListFed WL = null;
if (collWL != null)
{
    dkIterator iterWL = collWL.createIterator();
    while (iterWL.more() == true)
    {
        WL = (DKWorkListFed)iterWL.next();
        WL.retrieve();
        System.out.println("name = " + WL.getName() + " description = "
            + WL.getDescription());
    }
    iterWL = null;
}
```

C++

```
// ----- Call the listWorkLists function
DKSequentialCollection *collWL =
    (DKSequentialCollection*)svWF.listWorkLists();
DKWorkListFed *WL = NULL;
if (collWL != NULL)
{
    dkIterator *iterWL = collWL->createIterator();
    while (iterWL->more())
    {
        WL = (DKWorkListFed*)(void*)(*iterWL->next());
        WL->retrieve();
        cout << "name = " << WL->getName() << " description = "
            << WL->getDescription() << endl;
        cout << "Threshold = " << WL->getThreshold() << endl;
        delete WL;
    }
    delete iterWL;
}
delete collWL;
```

Accessing a worklist

You can access a worklist by creating an instance of `DKWorkListFed` that refers to the worklist which you created using the system administration client. The following example accesses a worklist named `WL0712` and displays the information contained in that worklist.

Java

```
// ----- Create the DKWorkListFed
DKWorkListFed WL = new DKWorkListFed(svWF, "WL0712");
WL.retrieve();
// ----- Display information about the worklist
System.out.println ("worklist name = " + WL.getName());
System.out.println ("description = " + WL.getDescription() +
    " owner = " + WL.getOwner() +
    " filter = " + WL.getFilter() +
    " threshold = " + WL.getThreshold() +
    " sort criteria = " + WL.getSortCriteria());
```

C++

```
// ----- Create the DKWorkListFed
DKWorkListFed* WL = new DKWorkListFed(svWF, "WL0712");
WL->retrieve();
// ----- Display information about the worklist
cout << "worklist name = " << WL->getName() << endl;
cout << "description = " << WL->getDescription() <<
    " owner = " << WL->getOwner() <<
    " filter = " << WL->getFilter() <<
    " threshold = " << WL->getThreshold() <<
    " sort criteria = " << WL->getSortCriteria() << endl;
// ----- Delete the worklist when you are done
delete WL;
```

Accessing work items

After you create the `DKWorkListFed`, you can retrieve the work items as a collection. The following example retrieves the work items.

Java

```
// ----- Create a collection and an iterator
DKSequentialCollection coll = (DKSequentialCollection)WL.listWorkItems();
dkIterator iter = (DKSequentialIterator) coll.createIterator ();
Object a;
DKWorkItemFed item;
String nodename;
String workflowname;

// ----- Step through the collections
while (iter.more ())
{
    a = iter.next ();
    item = (DKWorkItemFed) a;
    if (item != null)
    {
        item.retrieve ();
        nodename = item.name ();
        workflowname = item.workflowName ();
        System.out.println ("workitem node = " + nodename +
                             " workflow name = " + workflowname);
    }
}
iter = null;
```

C++

```
DKSequentialCollection *coll;
dkIterator *iter;
DKWorkItemFed* item;
DKString nodename;
DKString workflowname;
// ----- Create a collection and an iterator
coll = (DKSequentialCollection*)WL->listWorkItems();

if (coll != NULL)
{
    iter = coll->createIterator();
    cout << "listWorkItems" << endl;
    // ----- Step through the collections
    while (iter->more ())
    {
        item = (DKWorkItemFed*)((void*)(*iter->next()));

        if (item != NULL)
        {
            //item.retrieve ();
            nodename = item->name();
            workflowname = item->workFlowName();
            cout << "workitem node = " << nodename
                 << " workflow name = " << workflowname << endl;
            delete item;
        }
    }
    delete iter;
    delete coll;
}
```

Moving items in the workflow

As a workflow advances, you move work items from one activity to the next by using the `checkOut()` and `checkIn()` functions. The following example shows how to move the work items. Note that only the workflow user currently being assigned to perform the work item can check out and check in the work item.

Java

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

C++

```
DKWorkItemFed* item =new DKWorkItemFed(svWF, "wf1", "node1", wfuser);
item->retrieve();
// ----- Call the checkOut method to lock the workitem
item->checkOut();
// ----- Call the checkIn method
item->checkIn(NULL);
delete item;
```

Listing all the workflow templates

You can list all the workflow templates in a workflow service by calling the `listWorkFlowTemplates()` function. The following example lists the name and description of all the workflow templates in a workflow service referenced by the `DKWorkFlowServiceFed` object `svWF`.

Java

```
// ----- Call the listWorkFlowTemplates method
DKSequentialCollection collWT =
    (DKSequentialCollection)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed WT = null;
if (collWT != null)
{
    dkIterator iterWT = collWT.createIterator();
    while (iterWT.more() == true)
    {
        WT = (DKWorkFlowTemplateFed)iterWT.next();
        WT.retrieve();
        System.out.println("name = " + WT.name() + " description = "
            + WT.description());
    }
    iterWT = null;
}
```

C++

```
// ----- Call the listWorkFlowTemplates function
DKSequentialCollection *collWT =
    (DKSequentialCollection*)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed *WT = NULL;
if (collWT != NULL)
{
    dkIterator* iterWT = collWT->createIterator();
    while (iterWT->more())
    {
        WT = (DKWorkFlowTemplateFed*)(void*)(*iterWT->next());
        WT->retrieve();
        cout << "name = " << WT->name() << " description = "
            << WT->description() << endl;
        delete WT;
    }
    delete iterWT;
}
delete collWT;
```

Creating your own actions (Java only)

You can create your own actions that you can use in a workflow. You define the actions and add them to actions lists in DB2 Information Integrator for Content Administration. You create actions using action objects (`DKWorkFlowActionFed` objects). An action object is a meta data container that records detailed instructions about how a particular task is intended to be executed at the client node. Action objects (meta data containers) only record instructions; they do not initiate the invocation of the tasks that are described in the action meta data.

Actions can be grouped into an action list (`DKWorkFlowActionListFed`). A workflow container carries the name of the action list (not the contents of the

action list) in which a set of actions relating to the work item are associated. A client must retrieve the action list and then iterate through the entries (actions) in the list and react accordingly. The sample below shows you how to work with actions and action lists. The sample completes the following tasks:

1. Retrieves the work item.
2. Retrieves the container that is routed along with the work item.
3. Retrieves the action list from the container.
4. Gets the list of actions from the action list.
5. Starts the actions accordingly.

```
wit.retrieve(); // wit is a DKWorkItemFed object
DKWorkflowContainerFed wcn = wit.inContainer();
wcn.retrieve();
String alName = wcn.getActionList();
DKWorkflowActionListFed wal = new DKWorkflowActionListFed(dsFed);
wal.setName(alName);
wal.retrieve();
dkIterator iter = null;
if ((coll!=null) && (coll.cardinality()>0))
{
    iter = coll.createIterator();
    while (iter.more())
    {
        DKWorkflowActionFed act = (DKWorkflowActionFed) iter.next();
        System.out.println("ACTION = " + act.getCommand());
        Runtime.getRuntime().exec(act.getCommand());
    }
}
else
    System.out.println("NO ACTION DEFINED");
```

Working with the Information Integrator for Content workflow JavaBeans

This section describes the DB2 Information Integrator for Content workflow JavaBeans that you can use to connect to a Content Manager Version 8 server. The beans and exceptions are contained in the com.ibm.mm.beans.wcm package (see *Planning and Installing Information Integrator for Content* for more information). You can use the JavaBeans in builders that support them (see the product documentation for your builder for additional information).

The beans provide functionality for building workflow applications. For example, you can use the JavaBeans in your applications to start workflow processes that follow a predefined execution path predetermined by the process manager defined on your Content Manager server.

An example of a real-world use of the JavaBeans is automating an insurance process online, like submitting an accident claim. Using the JavaBeans, you can automate the entire process from when the claim gets submitted to when it gets closed.

Prerequisites

You must have the following components installed and configured before you can work with Information Integrator for Content Workflow JavaBeans:

- IBM WebSphere Application Server 4.0.4
- IBM Information Integrator for Content V 8.1, Fix Pack 1 and prerequisites

You must have installed Information Integrator for Content with the workflow option and an LDAP server to enable user import and user authentication.

- IBM Content Manager 8.1, fixpak 1 and prerequisites

See the *Planning and Installing Your Content Management System* manual for a list of supported LDAP server and for complete instructions for configuring your system to interact with one of the LDAP servers.

The userID selected as the DB2 UDB connection userID (the default is ICMCONCT) must be properly configured at the operating system level, and have the UserDB2TrustedConnect privilege set within your IBM Content Manager system.

- LDAP server supported by IBM Content Manager and WebSphere Application Server

Setting up the sample data model

To help you understand the steps needed to set up your data model, this section walks you through setting up the data model for a sample. The steps required to set up the sample data model below are the same general steps you follow to set up your own data model. You must complete the following steps using the DB2 Content Manager system administration client.

1. Create a privilege set called WCMPrivilegeSet and add the following privileges:
 - AllowConnectToLogon
 - ItemSQLSelect
 - ItemTypeQuery
 - ItemQuery
 - ItemAdd
 - ItemSetUserAttr
 - ItemSetSysAttr
 - ItemDelete
 - ItemMove
 - ItemLinkTo
 - ItemLinked
 - ItemAddLink
 - ItemRemoveLink
 - ItemCheckInOut
 - ItemAddToDomain
 - ItemGetWorkList
 - ItemGetWork
 - ItemRoute
 - ItemRouteStart
 - ItemRouteEnd
 - ItemUpdateWork
 - ItemGetAssignedWork
 - SystemDomainAdmin
 - SystemDomainQuery
 - SystemDefineUser
 - SystemGrantUserPrivs
 - SystemQueryUserPrivs
 - SystemDefineGroup
 - SystemQueryGroup
 - SystemDefinePrivs
 - SystemDefineDomain
 - SystemDefineACL
 - SystemDefineSemanticType

- SystemSetACL
- SystemDefineRM
- SystemDefineXdoObject
- SystemDefineSMSColl
- SystemSetReplicaRule
- SystemSetCtrlParm
- SystemQueryOtherDomains
- SystemDefineNLSLang
- SystemBatchCompileACL
- SystemDefineMimeType
- SystemManageKey
- SystemDefineAttrs
- SystemGetKey
- SystemDefineItemType
- SystemDefineNewKywdClass
- SystemQueryAllKywdClass
- SystemDefineLinkType
- SystemQueryItemType
- IKFAllPermissions
- IKFCreateCatalog
- IKFDeleteCatalog
- IKFRetrieveCatalog
- IKFUpdateCatalog
- IKFCreateCategory
- IKFRetrieveCategory
- IKFUpdateCategory
- IKFDeleteCategory
- IKFCreateTrainingDoc
- IKFRetrieveTrainingDoc
- IKFUpdateTrainingDoc
- IKFDeleteTrainingDoc
- IKFCreateRecord
- IKFRetrieveRecord
- IKFUpdateRecordIKFDeleteRecord
- IKFRunServerTask
- IKFRunAnalysisFunc
- ClientScan
- ClientPrint
- ClientImport
- ClientExport
- ClientSendMail
- ClientReceiveMail
- ClientReadBasePart
- ClientModifyBasePart
- ClientAddNewBasePart
- ClientDeleteBasePart
- ClientReadAnnotation
- ClientModifyAnnotation
- ClientReadNoteLog
- ClientAddToNoteLog
- ClientModifyNoteLog
- ClientReadHistory
- ClientAdvancedSearch
- ClientReadFolderContents
- WFWorklist
- EIPAdminServer

- EIPAdminEntity
- EIPAdminTextEntity
- EIPAdminTemplate
- EIPAdminInfoMining

For help with creating a privilege set, see the *Creating Privilege Sets* section of the Content Manager *System Administration Online Help*.

2. Create the user groups in the list below. For help with creating user groups, see the *Creating user groups* section of the DB2 Content Manager *System Administration Online Help*.

- Workflow Participants
- Content Publisher
- Project Lead
- Content Contributor
- Domain Expert

For now, do not add any users to the groups. You can add users to the groups in later steps.

3. Create the access control lists (ACLs) in the table below. Note that you should create a different ACL for each group of users that require the same access privileges. For help with creating ACLs, see the *Creating access control lists* section of the Content Manager *System Administration Online Help*.

Table 37. Access Control Lists

ACL name	Group	Privilege set
WCMACL	Workflow Participants	WCMPrivilegeSet
Content PublisherACL	Content Publisher	WCMPrivilegeSet
Content ContributorACL	Content Contributor	WCMPrivilegeSet
Project LeadACL	Project Lead	WCMPrivilegeSet

4. Create the attributes, with the characteristics shown, in the table below. Leave the default values if they are not specified below. See the *Managing object retrieval* and *Creating attributes* section of the Content Manager *System Administration Online Help* for additional information.

Table 38. Attributes

Name (case sensitive)	Attribute type	Minimum	Maximum
WCM_Fields	Variable Character	0	4,096
WorkPackageACL	Variable Character	0	500

Create the item types, with the characteristics shown, in the table below. Leave the default values if they are not specified below. See the *Managing object retrieval* and *Creating item types* sections of the Content Manager *System Administration Online Help* for help.

Table 39. Item types

Name (case sensitive)	Item type classification	Access Control	Attributes
WCM_Document	Item	WCMACL	WCM_Fields
WCM_Folder	Item	WCMACL	WorkPackageACL

Note: WCMACL represents an ACL that contains all workflow users.

5. Create a process manager. To create a process manager, you must designate work nodes and actions that define a new process. A work node is a step within a process at which items wait for actions to be taken by end users or applications, or through which items move automatically. You can define a one step process, or you can create one process with several steps within it.

Restriction: You must have at least one work node defined to create your process.

a. Create the work nodes

Create the work nodes, with the characteristics shown, in the table below. Leave the default values if they are not specified in the table. See the *Managing document routing* and *Creating workbaskets and collection points* sections of the *Content Manager System Administration Online Help* for more help.

Table 40. Work nodes

Name (case sensitive)	Description	Access Control List
Request Change	Initiate the change request	WCMACL
Make Change	Change the content	Content ContributorACL
Review Request	Review the change request	Project LeadACL
Review Change	Approve the change	Domain ExpertACL

b. Create a process manager

Create the process manager listed below with the characteristics shown. Leave the default values where they are not specified. See the *Managing document routing --> Defining a new process* section of the *System Administration Guide* for more help.

Name: Simple Change Process

Description: Process model for simple change process

Access control list: WCMACL

Table 41. Process manager

From node	Selection	To node
START	Continue	Request Change
Request Change	Continue	Review Request
Review Request	Accept	Make Change
Review Request	Reject	END
Make Change	Continue	Review Change
Review Change	Accept	END
Review Change	Reject	Make Change

6. Create users

You should create users on an LDAP server, such as IBM Directory Server and configure IBM WebSphere Application Server to use the LDAP server as its authentication mechanism in the **Security Center**. You must define users in the Content Manager server. See the *Creating users* section of the *System Administration Online Help* and follow the optional *Obtain from LDAP* instructions. WebSphere Application Server uses the LDAP uid attribute for authentication. By default, Content Manager uses the cn user attribute. For compatibility purposes, you should change the Content Manager LDAP configuration to use uid as the user attribute. You can also to set the cn and uid to the same value when creating the users in LDAP.

Create the following users on your LDAP server and then import them into Content Manager:

Table 42. Users for sample

User	Group
"Tara"	"Workflow Participants", "Content Publisher"
"Rob"	"Workflow Participants", "Project Lead"
"Greg"	"Workflow Participants", "Content Contributor"
"Dave"	"Workflow Participants", "Content Contributor"

7. (Optional) Adding user defined custom attributes to your workflow process manager

Below is example code that demonstrates how to add custom attributes to your workflow process manager:

```
java CustomAttributeTool -d datastore -u user -p password -w <worknode>
-n <attribute name> -v <attribute value> [...-n <attribute name> -v <attribute
value>]
```

Where

worknode - name of the work node to add the custom attribute(s) to. The work node must exist.

attribute name - the user defined name (this attribute need not exist already)

attribute value - the String value to be associated with attribute name.

Example:

```
java CustomAttributeTool -w Review Change -n WCM.Promote -v yes -n
WCM.Publish -v WCM Publish java AddCustomAttributes -h for more options.
```

You can download this tool from the IBM support Web site.

8. (Optional) Adding decision labels to your workflow process manager

Below is example code that demonstrates how to add decision labels to your workflow process manager.

```
java DecisionLabelTool -d datastore -u user -p password -w <worknode>
-l <decision label>
```

Where

worknode - name of the work node to add the decision label to.

The work node must exist.

decision label - text String to be shown as the decision label.

Example:

```
java DecisionLabelTool -w Review Change -l Was this change
implemented correctly?
java DecisionLabelTool -h for more options.
```

You can download this tool from the IBM support Web site.

9. Creating a work list

Create the work list below with the characteristics shown. Leave the default values if they are not specified. For additional help, see *Document Routing --> Creating worklists* section in the *System Administration Guide*.

Name: WCM_WL (case sensitive)

Access control list: WCMACL

Nodes: Add all the work nodes that are listed in the process managers' routes (Table 41 on page 399).

Optional sample setup:

Add the following work nodes from the **Node** tab: Request Change, Review Request, Make Change, Review Change

Using the workflow JavaBeans in your application

Before you call the beans, you must set the datastore name in the session. Begin using the beans by calling them from a servlet or JSP that runs on your WebSphere Application Server. Note that an `HttpServletRequest` object (this variable is called 'request' in the example code below) is associated with the servlet.

Example:

```
HttpSession session = request.getSession();
session.setAttribute("com.ibm.mm.beans.wcm.ICMServerName", "ICMNLSDDB");
```

In the example above, ICMNLSDDB is the datastore name.

The DB2 Information Integrator for Content workflow JavaBeans automatically login to the DB2 Content Manager server when you instantiate any of the workflow beans by calling the constructor. DB2 Content Manager uses the userID and password from the `LtpaToken` set by your WebSphere Application Server. The session is reused if you use the beans from different classes. You are automatically logged off when you terminate the session or the session expires.

Example code snippets

This section contains example code snippets for common tasks that you can complete using the DB2 Information Integrator for Content workflow JavaBeans. The code snippets are for your reference only and might not work in your application if they are used exactly as they appear below.

Creating a workflow process

This servlet code snippet creates and starts a workflow process based on the process manager 'Simple Change Process' with a user-defined parameter of 'subject'.

```
// the request variable has the com.ibm.mm.beans.wcm.ICMServerName
//set to ICMNLSDDB
WorklistHandlerAccessBean wb = new WorklistHandlerAccessBean(
    new WorklistHandlerKey(),request);

java.util.Hashtable myFields = new java.util.Hashtable();
myFields.put("subject","my subject");
String processName="My test job";
ExtendedActivityAccessBean activity =
    wb.createAndStartProcessAndClaimFirstActivity("Simple Change
    Process", processName, myFields);
```

Listing the workflow processes

```
//list all activities
Vector results = wb.getActivities("SELECT ACTIVITY, NAME, DESCRIPTION,
    STATE, OWNER, starttime from allactivities", -1);
//list only activities I can claim
Vector results = wb.getActivities("SELECT ACTIVITY, NAME, DESCRIPTION,
    STATE, OWNER, starttime FROM canclaim", -1);
```

Listing information about a workflow process

```
String activityId = "90 3 ICM8 ICMNLSDDB11 WORKPACKAGE58
    26 A1001001A02K06B34535J3113018 A02K21B14457G092121 13 204";
ExtendedActivityAccessBean activity = new ExtendedActivityAccessBean(new
    ExecutionObjectKey(activityId),null);
System.out.println("start date: " + activity.getStartDate());
System.out.println("unique job id: " +
    activity.getContainer().getJobId());
System.out.println("state: " + activity.getState());
System.out.println("Process Model: " + activity.getContainer().
    getManager().getName());
System.out.println("Process name: " +
```

```

        activity.getContainer().getName());
ExtendedDocumentAccessBean document =
        activity.getBinder().getMainDocument();
System.out.println("subject:" + document.getField("subject"));
System.out.println("potential owners: ");
java.util.Vector potentialOwners = activity.getPotentialOwners();
java.util.Enumeration e = potentialOwners.elements();
while (e.hasMoreElements()){
    String s = e.nextElement().toString();
    System.out.println(s);
}

```

Claiming and advancing a workflow process

```

// the user running this must have authority to claim at this worknode
String activityId = "90 3 ICM8 ICMNLSDB11 WORKPACKAGE58 26
A1001001A02K06B34535J3113018 A02K21B14457G092121 13 204";
ExtendedActivityAccessBean activity = new ExtendedActivityAccessBean(new
    ExecutionObjectKey(activityId),null);
activity.claim();
// completing the activity advances to the next work node in the routing
// process list decision choices

System.out.println("decisionChoices: ");
java.util.Vector v = activity.getDecisionChoices("Decision");
e = v.elements();
while (e.hasMoreElements()){
    String s = e.nextElement().toString();
    System.out.println(s);
}

// (assuming node has more than one decision choice for this example)
// make a decision
activity.setDecision("Decision","Accept");
activity.complete();

```

Setting and getting fields in a workflow process

```

Setting and getting fields in a workflow process
String activityId = "90 3 ICM8 icmnlsdb11 WORKPACKAGE58 26
A1001001A02I30B32323A1048018 A02J04A60100A986431 13 204";
ExtendedActivityAccessBean activity = new ExtendedActivityAccessBean(new
    ExecutionObjectKey(activityId),null);
BinderAccessBean binder = activity.getBinder();

// getMainDocument()
ExtendedDocumentAccessBean document = binder.getMainDocument();
document.updateField("my field","my field test");
System.out.println("Successfully updated.");

// getField(String fieldName)
String fieldValue = document.getField("my field");
System.out.println("my field =" + fieldValue);

```

Chapter 10. Building applications with non-visual and visual JavaBeans

This chapter describes the non-visual and visual JavaBeans provided in Information Integrator for Content.

The Information Integrator for Content JavaBeans can be divided into the following categories:

Non-visual beans

You can use the non-visual beans to build Java and Web client applications that require a customized user interface. The non-visual beans support the standard bean programming model by providing default constructors, properties, events and a serializable interface. You can use the non-visual beans in builder tools that support introspection.

Visual beans

The visual beans are customizable, Swing-based, graphical user interface components. Use the visual beans to build Java applications for Windows. You can place them within windows and dialogs of Java-based applications. Because the visual beans are built using the non-visual beans (as a data model), you must use them in conjunction with the non-visual beans when building an application.

Understanding basic beans concepts

JavaBeans (hereinafter referred to as beans) are reusable software components that are written in the Java programming language and can be manipulated using builder tools that are beans-aware. Because the beans are reusable, you can use them to construct more complex components, build new applications, or add functionality to existing applications. You can do all of this visually, using a builder, or manually, by calling the beans methods from a program.

Beans are Java classes that adhere to specific conventions regarding property and event interface definitions. By conforming to the conventions, you can turn almost any existing programming component or Java class into a bean.

Beans define a design-time interface that allows application designer tools, or builder tools, to query components to determine the kinds of properties these components define and the kinds of events they generate or to which they respond. You do not have to use special introspection and construction tools when working with beans. The pattern signatures are well defined and can be easily recognized and understood by visual inspection.

Beans have the following characteristics:

Introspection

Introspection is the process by which a builder tool determines and analyzes how a bean works at design and run time. Because the beans are coded with predefined patterns for their method signatures and class definitions, tools that recognize these patterns can "look inside" a bean and determine its properties and behavior. Each bean has a related bean information class, which provides property, method, and event information

about the bean itself. Each bean information class implements a `BeanInfo` interface, which explicitly lists the bean features that will be exposed to application builder tools.

Properties

Properties control a bean's appearance and behavior. Builder tools introspect on a bean to discover its properties and to expose those properties for manipulation. This allows you to change a bean's property at design time.

Customization

The exposed properties of a bean can be customized at design time. Customization allows you to alter the appearance and behavior of a bean. Beans support customization by using property editors or by using special, sophisticated bean customizers.

Events

Beans use the Java event model to communicate with other beans. Beans can fire events. When a bean fires an event it is considered a source bean. A bean can also receive an event, in which case it is considered a listener bean. A listener bean registers its interest in the event with the source bean. Builder tools use introspection to determine those events that a bean sends and those events that it receives.

Persistence

Beans use Java object serialization, by implementing the `java.io.Serializable` interface, to save and restore states that might have changed as a result of customization. For example, the state is saved when an application customizes a bean in an application builder, so that the changed properties can be restored at a later time.

Methods

All bean methods are identical to methods of other Java classes. Bean methods can be called by other beans or through scripting languages.

Using JavaBeans in builders

This section explains how to use JavaBeans in IBM Websphere Studio Application Developer, and in other builders.

To use builders other than IBM Websphere Studio Application Developer, make sure that the builder supports Java 2. Follow the builder's instructions for adding new jar files to add the jars specified in the instructions below. Then, follow the builder's instructions for adding beans from a jar to add the Information Integrator for Content beans, which are in `cmb81.jar`.

Important: To use the beans, you must have at least Java JDK 1.4 or higher.

The `Samples` directory contains code samples of the non-visual beans.

Using IBM Websphere Studio Application Developer

You can use the non-visual beans to build servlets and JSP pages in Webphere Studio Application Developer by completing the following steps:

1. Create a Web project for your Web application.
2. In the properties for the project, in `Java Build Path` | `Libraries`, specify the following JAR files:

`%IBMCMROOT%\lib\cmb81.jar`

```
%IBMCMROOT%\lib\cmbview81.jar
%IBMCMROOT%\lib\cmbsdk81.jar
\SQLLIB\java\db2java.zip
```

3. If you plan on using the Information Integrator for Content servlets and JSP taglib, you must also specify the following files:

```
%IBMCMROOT%\lib\cmbservlet81.jar
%IBMCMROOT%\lib\cmbtag81.jar
```

For the tag library, you also need to import the taglib.tld file for Information Integrator for Content's JSP taglib into your web application:

- Copy \IBMCMROOT%\lib\taglib.tld to the webApplication\WEB-INF directory in your web application.
- Configure the taglib in the webApplication\WEB-INF\web.xml file in you web application by adding the following:

```
<taglib>
                                <taglib-uri>cmb</taglib-uri>
                                <taglib-location>/WEB-INF/taglib.tld</taglib-location>
</taglib>
```

4. Since the JARs listed above contain J2EE classes, you need to include the J2EE JAR, usually located in: \Program Files\IBM\Application Developer\plugins\com.ibm.etools.websphere.runtime\lib\j2ee.jar

Invoking the Information Integrator for Content JavaBeans

The beans in the Information Integrator for Content layer can be called in one of two ways. You can call them directly by using their public interfaces (public methods). In this case, explicit Java exceptions are thrown to indicate error events.

Another method for calling the functionality on the session-wide beans is to wire any instances of this type of bean to other Information Integrator for Content beans using request and reply events. When using this method, remember the following:

- The CMBConnection bean listens to connection request events and replies by firing connection reply events.
- The CMBDataManagement beans listens to data request events and fires data reply events in return.
- The CMBSchemaManagement beans listens to schema request events and fires schema reply events in return.
- The CMBQueryService bean listens to search request events and fires search reply events in return.
- The CMBWorkflowDataManagement beans listens to workflow data request events and fires workflow data reply events in return.
- The CMBWorkflowQueryService beans listens to work list request events and fires work list reply events in return.

Working with the non-visual beans

Information Integrator for Content provides a set of non-visual JavaBeans that you can use to build Java applications. The non-visual beans are a set of Java classes that follow the beans conventions. They are built using the Information Integrator for Content and DB2 Content Manager Java connector classes. You can use the beans to build Servlets or Java Server Pages (JSP), although the beans can also be used in a command line or in Windows applications.

Following is a list of the benefits of using the non-visual beans:

- Provide a federated access mechanism and common programming model for the many different connectors that ship with Information Integrator for Content.
- Allow you to program at a higher level of abstraction.
- Hide the complexity and details of individual connectors.
- Allow you to leverage beans support built into most commercial development environments.
- Make it easier for your application to use multiple connectors, or to migrate, without having to make big changes to your application.

Using the beans makes building basic applications easier, however, there are some limitations when using the beans. The following is a list of functionality that the beans do not provide:

- Administrative and configuration functionality.
- Batch import. The beans only support single item type import capabilities. If you need batch processes for importing and exporting large amounts of data, you should use the connector interfaces.
- Functionality that is server specific. For example, the functionality related to sub-items within items is only available in the Content Manager Version 8 connector only.

The limitations listed above can, in some cases, be bridged by using accessor methods that allow access to the underlying Java API's.

Important: The Information Integrator for Content beans are not Enterprise JavaBeans (EJB). Therefore, they cannot be hosted directly inside the managed environment provided by containers like IBM Websphere. However, they can be used from inside EJBs as the underlying connection mechanism to unstructured data repositories.

Non-visual bean configurations

Non-visual beans have local, remote and dynamic configurations. The configuration, however, is completed at the connector level and the beans pass through these settings to the connector.

local Connects directly to the content server.

remote
Connects to a content server using an RMI server.

dynamic
Enables an application that dynamically switches between local and remote based on the `cmbcs.ini` file. The `cmbcs.ini` file specifies whether the content server is local or remote.

Understanding the non-visual beans features

You can use the Information Integrator for Content beans in JSP since their properties are typically simple types like strings and arrays. In essence, they act as the model component for Web applications because they are modeled using the Model View Controller (MVC) design pattern. Note that the view component is typically comprised of JSP and the controller component of servlets (like the ones in the EJB servlet kit). The following is a list of the Information Integrator for Content non-visual beans features:

- Provide access to the schema definitions in the library server.

- Provide CRUD (create, retrieve, update, delete) methods for documents, simple (non-resource) items, and resource items in all of the repositories supported by Information Integrator for Content.
- Provide functionality to search and retrieve documents, simple items, and resource items in all of the repositories supported by Information Integrator for Content.
- Support conversion of data types to viewable formats.
- Act as a federating layer that enforces a consistent set of semantics across the many content management repositories supported by Information Integrator for Content.
- Integrate and expose the functionality provided in the Information Integrator for Content workflow services.
- Provide document extraction and conversion services as well as support for managing document annotations.
- Provide sorting and conversion functionality.
- Provide events that are fired for key actions occurring on the constituent beans, like connect and disconnect events, search results notification events, and content change notification events.

Non-visual beans categories

The non-visual beans can be divided into the following categories:

Datastore beans

These beans exist across a typical user session and present specialized services to the user. The session-wide beans include the following:

CMBConnection

This bean maintains the connection to a backend server which could be a native content server or a federated server. This bean is required in order to use any of the JavaBeans.

CMBSchemaManagement

Used to work with repository metadata.

CMBDataManagement

Used to work with repository data.

CMBQueryService and CMBSearchResults

Used to run queries and work with the results from the queries.

CMBWorkflowDataManagement and CMBWorkflowQueryService

Used to work with Information Integrator for Content advanced workflow processes.

CMBDocRoutingDataManagement and CMBDocRoutingQueryService

Used to work with CM V8 document routing processes.

CMBDocumentServices

Used to provide document conversion, document manipulation, and annotation services.

Helper beans

The helper beans exist in the context of one or more of the session-wide beans and are primarily used for encapsulation of data values and for providing services to the session-wide beans. The helper beans available include the following:

CMBEntity

Represents data item definitions available in content management

repositories. For example, for a CM V8 repository, a CMBEntity represents both item types and child component definitions, while for a CM V7 repository, a CMBEntity represents an index class. CMBEntity is a helper class for CMBSchemaManagement.

CMBAttribute

Represents attribute definitions in the repositories. CMBAttribute is a helper class for CMBSchemaManagement.

CMBSearchTemplate

Represents a federated search template. CMBSearchTemplate is a helper class for CMBSchemaManagement.

CMBSTCriterion

Represents a search criterion that is a part of a federated search template. CMBSTCriterion is a helper class for CMBSchemaManagement.

CMBItem

Represents instances of documents, resource items and non-resource items. CMBItem is a helper class for CMBDDataManagement.

CMBObject

Represents instances of resource items, base parts and notelog parts. CMBObject is also used to represent BLOB attributes. CMBObject is a helper class for CMBDDataManagement.

CMBAnnotation

Represents instances of annotation parts for CM V8 repositories and notes for OnDemand repositories.

CMBPrivilege

Provides the functionality required to retrieve privilege related information from a CM or Information Integrator for Content supported repository.

Ancillary beans

The ancillary beans are not essential in applications, but can be useful for enhancing the functionality. Following is a list of the ancillary beans.

CMBConnectionPool

Used to provide pooling services to CMBConnection beans. The Java API class DKDatastorePool is used to maintain the DKDatastore instances that are used by CMBConnection instances. By moving the pooling to the Java API level, the content servers can be better managed, since they can be pooled more intelligently based on the type of server.

CMBUserManagement

Used in connections to federated repositories to manage the mappings of federated users to native server users.

CMBExceptionSupport

Provides a framework for common exception event handling.

CMBTraceLog

Provides a common trace event-handling framework and provides listener capabilities for trace events fired by other beans.

XML services bean and helper classes

The XML bean takes messages in the form of XML requests and replies to

perform various operations on DB2 Content Manager, including searching, creating, updating, batching, exporting items into XML format, export item type definitions as XML schemas, and workflow operations. The XML bean leverages the XML support in the API for item import, item export, and schema export operations.

CMBXMLServices

Constructor for the XML services bean.

CMBXMLMessage

Used as a wrapper class for an XML document to describe a request or reply to the beans. It contains both the XML source of the request document, and the set of attachments associated with the message. The XML document may be a file, string, input stream, or document object model (DOM).

CMBXMLAttachment

Used together with the CMBXMLMessage class to represent an attachment in the XML message. This object maps to a resource object or a document part.

Workflow beans

The workflow beans provide workflow services. The workflow services provided by the Information Integrator for Content beans layer support two types of workflow systems: advanced workflow and document routing. Advanced Workflow functionality uses MQSeries® Workflow, while document routing is a workflow system integrated into the Content Manager Version 8 product and API set. The beans layer provides the full set of objects required to create workflow definitions, execute workflow instances based on the created definitions, and manage instances of executing workflows. The following beans are the main components of the advanced workflow support:

CMBWorkflowDataManagement

Use this bean to create and work with advanced workflow instances. An instance of this bean can be retrieved from the CMBConnection object. This bean provides the following functionality:

- Starting, terminating, suspending, and resuming a workflow instance.
- Transferring work-items and work notifications from one user to another.
- Canceling work notifications.

CMBWorkflowQueryService

The CMBWorkflowQueryService provides an interface for querying advanced workflow related information. An instance of this bean can be retrieved from the CMBConnection object. This bean provides support for retrieving the following information:

- Information on workflows in the system.
- Information on the work items moving through the system as part of active workflows.
- Work list related information.
- Information on all the registered work notifications.

The following beans are the main components of the document routing support.

CMBDocRoutingManagementICM

Use this bean to create and manage document routing processes. You can obtain an instance of this bean from an instance of the `CMBConnection` object. This bean provides support for the following features:

- Starting, terminating, suspending and resuming a document routing instance.
- Checking out a CM item contained inside a work package.
- Setting properties for work packages being routed by a document routing instance.

CMBDocRoutingQueryServiceICM

This bean provides an interface for querying information related to document routing processes. You can obtain an instance of this bean from an instance of the `CMBConnection` object. This bean provides support for retrieving the following information:

- Information relating to the work packages being routed through the document routing system.
- Information relating to the processes that are currently active in the system.
- Information on all the work lists that are present in the system.
- All the work nodes that are part of the system.

Document Services beans

The document services beans provide document streaming and document annotation services. The following beans and classes are from the Java viewer toolkit:

CMBDocumentServices

The `CMBDocumentServices` bean provides services needed when working with documents, including the functionality needed to render, convert, and reconstitute the pages of one or more documents.

CMBDocument

This class represents the entity created when loading a document using `CMBDocumentServices`. In essence, `CMBDocument` is a container for the pages in the document. `CMBDocument` also allows you to query and set properties that control a document's characteristics.

CMBPage

This class provides a representation of a particular page in a document. The functionality in this class allows you to specify and control the properties of a set of renderable images that can be generated for the page.

CMBPageAnnotation

This class models an annotation that can be associated with a page in a document. All supported annotations are modeled by sub-classes of this bean. Also, the `CMBPageAnnotation` itself contains properties like the page the annotation is on and the annotation type. The sub-classes include the following:

- `CMBArrowAnnotation`
- `CMBCircleAnnotation`
- `CMBHighlightAnnotation`

- CMBLineAnnotation
- CMBNoteAnnotation
- CMBPenAnnotation
- CMBRectAnnotation
- CMBStampAnnotation
- CMBTextAnnotation

Other beans classes

The bean classes listed in this section provide a variety of functionality, as described below.

BeanInfo Classes

Allow for explicit exposure of the features of the Information Integrator for Content beans in a separate, associated class that implements the BeanInfo interface.

Exception classes

Used to encapsulate exceptions in the beans layer. All the exception classes inherit from the base class CMBException. Each of the sub-classes of CMBException indicates a specific error condition. In each case, properties of the exception object can be used to obtain detailed error information on the error condition that led to the exception being thrown. When the beans are used in the event-driven fashion, exceptions are set as events.

Event and listener classes

Implement the standard beans event listener model. Classes have to explicitly request an event by implementing the listener interface associated with the event and by registering that listener interface with the object that generates an event. The Information Integrator for Content beans layer provides event and listener pairs for schema access operations, data access operations, workflow operations and search operations.

Session listeners

These are the listener classes that exist at the session-wide level. In the Information Integrator for Content beans, there are currently session listeners that track connection requests and connection reply events.

Considerations when using the non-visual beans

You can use the non-visual beans to enable general-purpose applications with the functionality required to access content management repositories supported by Information Integrator for Content. This section contains some tips on specific usage patterns in the beans.

Singletons in the beans

CMBConnection has methods to obtain access to instances of the other session-wide Information Integrator for Content beans. When the session-wide beans like CMBSchemaManagement and CMBDataManagement are obtained in this way, they are already wired to the CMBConnection bean (from which they are obtained) to be informed of a connection or a disconnection, and to share trace and exception event handlers. Only a single instance of each of the other session-wide beans is created. If these methods are called repeatedly, the same instance is returned (singleton design pattern). If session-wide beans are created in the application, and not by the CMBConnection bean, they must be wired to a CMBConnection bean to be used.

Threading considerations in the beans

A single instance of the CMBConnection bean can only be used on a single thread at any point in time. This restriction extends to all other beans that are associated to a CMBConnection bean (through the connection property of the associated bean). That means that you must create separate connections for each thread. Alternatively, multiple threads can obtain and free connections using the CMBConnectionPool bean. Therefore, each thread should obtain, use, and free a connection.

All the session-wide beans have affinities to an instance of the CMBConnection from which they were retrieved or with which they have been associated after creation. This implies that an instance of the session-wide beans such as CMBSchemaManagement can only be used by a single thread at any given time. If the session-wide bean instance is used by multiple threads, you must perform explicit synchronization in your application to ensure that only a single thread is actively using the session-wide bean instance at any given time.

All session-wide beans also listen to connection reply events generated by the CMBConnection beans. This allows them to recognize that the underlying content repository with which the CMBConnection bean instance is associated has changed, so that the beans can take appropriate action.

Unlike the CMBConnection, the CMBConnectionPool bean is designed for multithreaded use. Multiple threads can simultaneously call the methods related to obtaining and freeing connection objects. Any connection obtained from the pool is an instance of CMBConnection and is restricted to single-thread access. Any connection obtained from the connection pool bean should be returned to the pool as soon as possible after its use, so that it may be made available to other threads that might be requesting connections from the pool.

Changing locale in display names

Display names are alternate names that you can assign to entities and attributes to make them more descriptive. For example, you can assign a display name of Insurance documents to an item type called Doc 26 (Doc 26 would be considered the non-display name).

In the DB2 Content Manager Version 8.3 connector, you can now translate display names for an entire set of languages instead of just one. To toggle between the languages, you would call the setLocale API within the CMBConnection object:

```
void setLocale(java.util.Locale locale);  
java.util.Locale getLocale();
```

where the locale class variable must contain the three-letter machine locale.

Both CMBEntity and CMBAAttribute contain three methods that control whether to return display names or not:

getNonDisplayName()

Returns the real, non-display name.

getDisplayName()

Returns the display name translated in your machine's locale.

getName()

If displayNamesEnabled equals false (this is the default), then getName() returns the non-display name. If displayNamesEnabled is set to true, then getName() returns the display name (translated in your machine's locale).

Tracing and logging in the beans

You can enable tracing on all session-wide beans in the Information Integrator for Content beans layer. Enabling tracing on an instance of the CMBConnection bean also enables tracing on any Information Integrator for Content bean obtained from this connection bean, including schema management, data management, query service, and workflow beans.

When tracing is enabled, tracing events are fired. A utility bean, CMBTraceLog can listen to trace events and write trace records to a defined log, stdout, stderr, or window (when used with the visual beans).

All session-wide beans also listen to tracing events. The tracing functionality in the beans writes logging information to the same log file as the Java API if log4j is used for logging.

Understanding properties and events for non-visual beans

Each non-visual bean provides the following:

- Imported properties
The property value is determined by other beans at run time by PropertyChange or VetoableChange events. Beans that have import properties must listen to PropertyChange or VetoableChange events.
- Exported properties, vetoable or not
A non-visual bean may have a constrained property and some other beans might have interest in its value. Whenever its value is changed, the bean is responsible for generating a PropertyChange or VetoableChange event.
- Stand-alone properties
No other beans have interest in this property value.
- Events generated by this bean
- Events in which this bean is interested

Building an application using non-visual beans

A sample non-Graphical User Interface (GUI) application

The example in this section uses non-visual beans to create a sample non-GUI application. The sample application includes every bean except the CMBUserManagement bean. The complete sample application from which this example was taken (DemoSimpleApp1.java) is available in the Samples directory. The sample application shows how to:

1. Connect to the Information Integrator for Content (federated) server
2. Get a list of search template names
3. Use the search template name to get a list of search criteria names
4. Select a search template and gets its search criteria
5. Complete the search values and submits a query
6. Print the result using the search results bean
7. Select a result row and displays it
8. Disconnect from the server

Working with visual beans

Visual beans allow you to integrate the functionality of DB2 Information Integrator for Content 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 Information Integrator for Content visual beans must also contain an instance of the `CMBConnection` nonvisual bean.

CMBLogonPanel

This bean displays a panel to login to DB2 Information Integrator for Content or to content servers such as DB2 Content Manager Version 8.3. 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 415) displays a window that lets users login to a content server, update user mappings, and change a password.

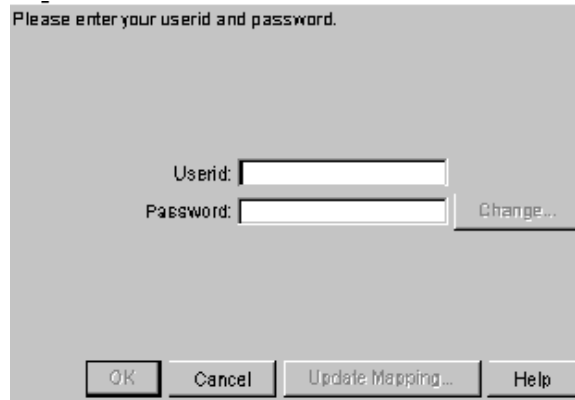


Figure 20. CMBLogonPanel bean window

In the CMBLogonPanel bean, when a user clicks **Change**, the **Change Password** window appears (see Figure 21.) The user enters the old password, and enters the new password twice.

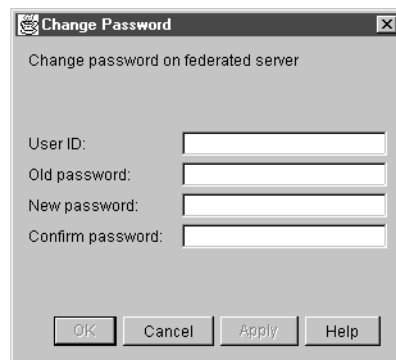


Figure 21. Change Password window

In the CMBLogonPanel bean, when a user clicks **Update Mapping** in the Logon window, the **Update Userid Mapping** window is displayed (see Figure 22 on page 416). 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 DB2 Information Integrator for Content federated database.

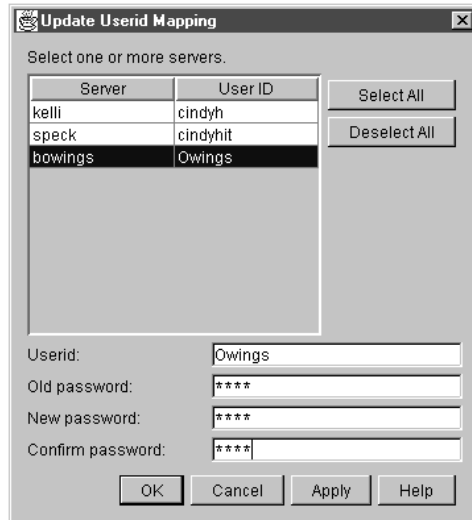


Figure 22. Update Userid Mapping window

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 417 shows the simple template list style. Figure 25 on page 417 shows the drop-down template list style.

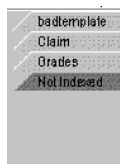


Figure 23. Image template list style

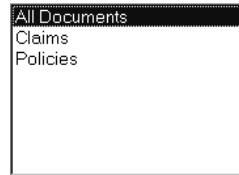


Figure 24. Simple template list style



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 Application Programming 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.

Figure 27. CMBSearchPanel Bean

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 28 shows the CMBSearchResultsViewer bean with the **Search Results** folder selected.

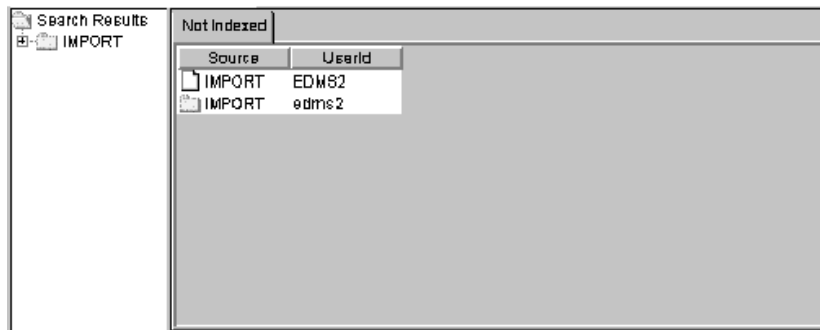


Figure 28. CMBSearchResultsViewer bean

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 29 on page 420 shows the tree and details panes of the `CMBFolderViewer` bean.

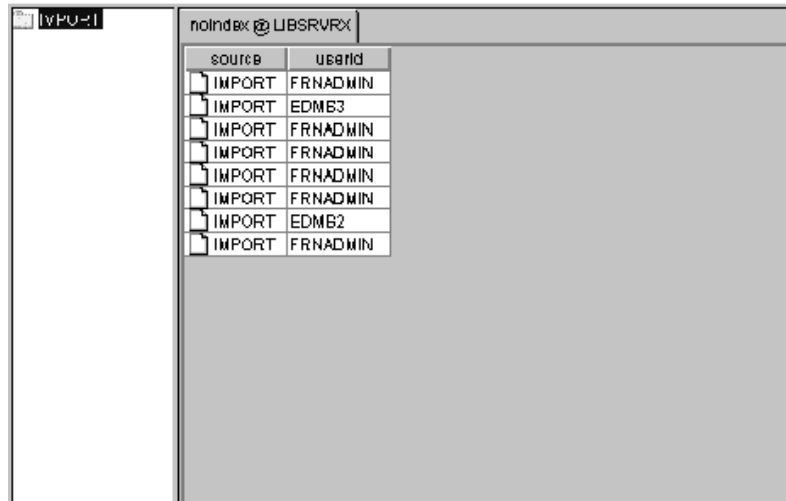


Figure 29. CMBFolderViewer bean

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 might 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.

Automobile Policy Declarations

XYZ Insurance Company
 Greenhill Ave
 Mt Town, CA
 (838) 838 888 Page 1 of 1

Declaration Type:		Insured Last Name	Insured First Name	
Renewal		Edward	Gregory	
Policy Type:		Insured Last Name	Insured First Name	
Member				
Policy Number:		Street		
10452136412		255 Green Rd.		
Policy Period From	Policy Period To	City	State	Zip
02/13/2001	02/13/2002	Ocean	CA	90062
Process Date	Insured Since	Home Phone Number	Office Phone Number	
01/10/2001	1997	(408) 972-2596	(408) 474-0674	

Drivers				
1	2	3	4	5
Gregory				

Vehicles				
Item	Make	Model Year	Body Type	VIN
01	Toyota	1996	4D Sedan	1HGM210H4KL087268
02	Nissan	1999	4C SUV	3C5A355R4KL987500

Figure 30. CMBDocumentViewer Bean

Viewer specifications

There are two ways to specify viewers:

1. In the system administration client, specify the viewers using the MIME Type to Application Association Editor, by selecting the **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 Information Integrator for Content Administration and using the Mime2App property, the specification using the Mime2App will take precedence.

Default viewers

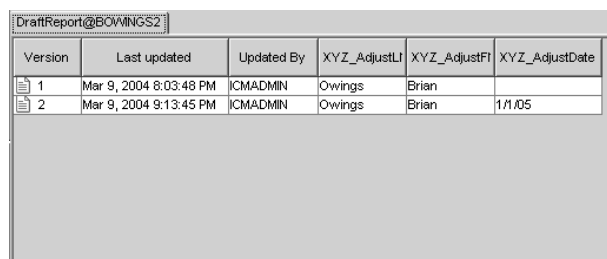
If no viewer is specified for a particular content type, a default viewer will be launched. For documents from OnDemand, the OnDemand client (in view-only mode) is launched if installed. Documents from all other content servers will be viewed using the CMBGenericDocViewer from the Java viewer toolkit. To edit annotations, select "Edit Document" from the "File" menu of the viewer.

Launching external viewers

Use the Mime2App property of CMBDocumentViewer to specify applications to launch as document viewers for documents of certain MIME types. Use setMime2App with a properties object as the argument that has names of MIME types mapping to values that are executable names.

CMBItemAttributesEditor bean

The CMBItemAttributesEditor bean (see Figure 31) displays a window for viewing and modifying the entity (item type) and attributes of a folder or document.



The screenshot shows a window titled "DraftReport@BOWINGS2". Inside, there is a table with the following data:

Version	Last updated	Updated By	XYZ_AdjustL	XYZ_AdjustFI	XYZ_AdjustDate
1	Mar 9, 2004 8:03:48 PM	ICMADMIN	Owings	Brian	
2	Mar 9, 2004 9:13:45 PM	ICMADMIN	Owings	Brian	1/1/05

Figure 31. CMBItemAttributesEditor bean

A list containing all available entities appears at the top of the window. The current entity is selected by default. A list of attributes for that entity appears beneath the entity. The text fields (first name, last name, and so forth) initially contain the current values for the item.

If users select a new entity, any attributes with the same names as the previously-selected entity have their values propagated to the like-named attributes in the new entity.

Clicking **Reset** returns the entity and attributes to their original values.

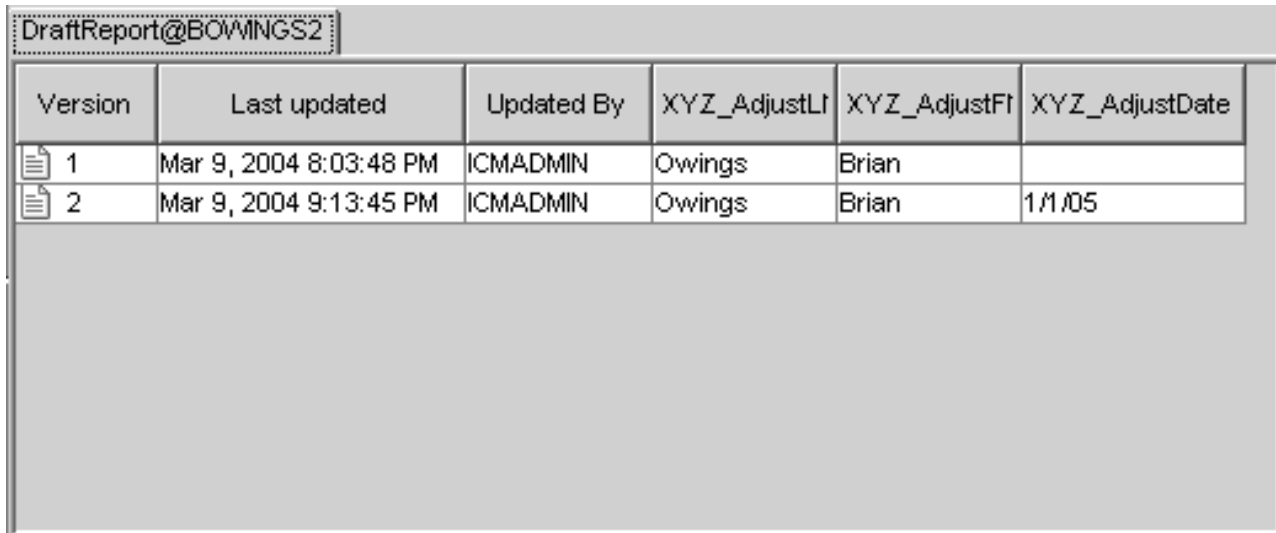
Clicking **OK** updates the entity and attributes and triggers events before and after the update. You can use the event before the update to validate fields or complete missing fields before the update is performed. This event can veto the specified update.

Vetoing changes in the **CMBItemAttributesEditor**

You can provide additional validation logic to the **CMBItemAttributesEditor** that verifies attribute values entered by the user and modifies them, or rejects an update, if the values are not valid. Do this by providing a handler for the **CMBEditRequestedEvent**.

CMBVersionsViewer bean

The **CMBVersionsViewer** bean displays a table of versioning attributes for a single document or item. The versioning attributes displayed are: the version number, the user ID of the creator, the timestamp when the version was created, the user ID of the latest updater, and the timestamp of the last update. From the versions viewer, you can view the different versions of an item or update the attributes of an item.





Version	Last updated	Updated By	XYZ_AdjustLt	XYZ_AdjustFl	XYZ_AdjustDate
 1	Mar 9, 2004 8:03:48 PM	ICMADMIN	Owings	Brian	
 2	Mar 9, 2004 9:13:45 PM	ICMADMIN	Owings	Brian	1/1/05

Figure 32. *CMBVersions Viewer* bean

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 **CMBCConnection** non-visual bean. You must set the **Connection** property for the visual bean to operate correctly.

CollationStrength

All beans that perform sorting have a **CollationStrength** property. The values defined for **CollationStrength** property are the same values defined for the `java.text.Collator` class of Java.

Hiding/Showing buttons

You can hide or show the buttons that appear on all visual beans. Use the *setNameButtonVisible* property, where *Name* is the name of the button.

Save/restore configuration

The CMBSearchTemplateViewer, CMBSearchResultsViewer, and CMBFolderViewer have two methods - *loadConfiguration* and *saveConfiguration*- that can be used to save and restore field values and column sizes between application sessions. A properties object is an argument for all these methods. You can use the same properties object for all three beans. The names of the saved properties are unique across the beans.

Support for child component attributes

The visual beans support a single level (not nested) child components. This is similar to the Windows client. For details, see the Windows client documentation.

Help events

Each visual bean generates a CMBHelp event when the user requests help, either by clicking the **Help** button or pressing F1. Some beans generate the following help-related events when users press F1 or Help from secondary windows:

CMBChangePasswordHelpEvent

When **Help** is clicked on the Change Password window

CMBUpdateMappingHelpEvent

When **Help** is clicked on the Update Mapping window

CMBLoginFailedHelpEvent

When **Help** is clicked on the Server Logon Failed window

CMBServerUnavailableHelpEvent

When **Help** is clicked on the Server Unavailable window

Tip: One possible method of handling help from all of these sources is to create a single class that implements the listeners for all of these events. Within the *onHelp* method, additional logic might be needed to determine which bean was the source of the event, and display help text appropriate for that bean.

Replacing a visual bean

It is possible to replace one of the visual beans with another bean or with Swing components. To do this, the new bean should implement the handlers for the events of the visual bean it is replacing. It should also generate at least the key events of the bean it is replacing. The key events are described in Table 43.

Table 43. Visual beans and key events

Visual bean	Key events
CMBSearchTemplateList	CMBTemplateSelectedEvent
CMBSearchTemplate Viewer	CMBSearchStartedEventCMBSearchResults Event
CMBSearchResultsViewer	CMBViewDocumentEvent CMBViewFolderEvent-CMBEditItemAttributesEvent
CMBFolderViewer	CMBViewDocumentEvent CMBEditItem AttributesEvent
CMBDocumentViewer	CMBDocumentOpenedEvent CMBDocument Closed Event
CMBItemAttributesEditor	none

All data needed for implementing the bean function is available either from events that the bean is handling or from the CMBConnection non-visual bean.

Building an application using visual beans

A sample client application that was written using the visual beans is provided for you. The source files for the sample are located in: samples. You should also read the readme.html file in this directory for details about the sample client and setup requirements.

The following sections show how the visual beans fit together when you build an application.

Connecting the visual beans

This section explains one scenario for connecting visual beans to create a simple application. Except for the **Search** button, all beans are connected by adding the target bean as a listener of the indicated event of the source bean. For example, to connect the SearchTemplateList to the SearchTemplateViewer, a single line of code is needed. To add a button for launching searches, use a standard JButton. Create an inner class to cause the action event from the button to invoke the appropriate method.

In Figure 33, the lines from each of the beans to the connection bean indicate that the bean contains a reference to the connection bean. This is created by setting the connection property for each bean. For example, to create a reference from the logon panel bean to the connection bean, a line of code is needed.

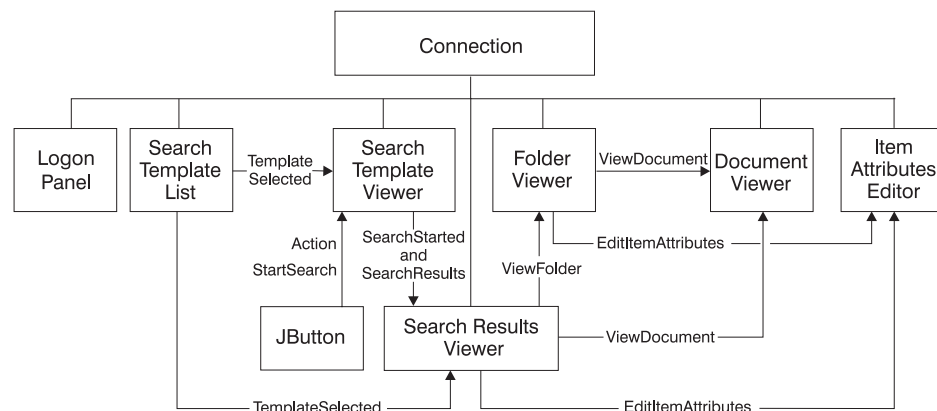


Figure 33. Visual bean connections

Figure 33 shows nine beans. A JFrame or other container bean would be the parent of all of these beans. One possible order of events during run time might be:

1. The user enters a user ID and password into the logon window and clicks **OK**. The CMBLogonPanel bean invokes the connect method of the CMBConnection bean to establish the connection to the server.
2. The connection bean establishes the connection. The CMBSearchTemplateList bean retrieves and displays the list of search templates for that user ID. (No methods need to be invoked to cause this to happen. The CMBSearchTemplateList bean is listening to the appropriate events of the CMBConnection bean. CMBSearchTemplateList sets up the listeners when a CMBConnection bean associated itself with it using the setConnection method.)

3. The user selects a search template from the list. The `CMBSearchTemplateList` bean generates a `CMBTemplateSelectedEvent`. Both the `CMBSearchTemplateViewer` and the `CMBSearchResultsViewer` are listening for the event. The `CMBSearchTemplateViewer` displays the appropriate template. The `CMBSearchResultsViewer` clears and displays columns in the details pane as defined by the template.
4. The user completes the template, and either presses Enter or clicks **Search**. If the user clicks **Search**, the action event handler invokes the `startSearch` method. If the user presses Enter, the `startSearch` method is invoked implicitly.
5. The `CMBSearchTemplateViewer` bean validates the template fields to determine whether a search can begin. If the search can begin, a `CMBSearchStartedEvent` is generated. `CMBSearchResultsViewer` listens for a `CMBSearchStartedEvent` and clears the results in preparation for new search results.
6. As the search progresses, `CMBSearchResultsEvents` are generated to provide partial search results to the `CMBSearchResultsViewer`. (When the search is completed the `CMBSearchCompleted` event is generated. This event can be used to enable the **Search** button again if it was disabled at the start of the search.)
7. The user can expand folders in the Search Results window, then select a document or folder for viewing. When this is done, a `CMBViewFolderEvent` or `CMBViewDocumentEvent` is generated. The `CMBFolderViewer` and `CMBDocumentViewer` beans are listening to their respective events, and display the folder or document.
8. From the `CMBFolderViewer`, users can select a document to view. Selecting a document for viewing generates a `CMBViewDocumentEvent`. The `CMBDocumentViewer` listens for this event and displays the document in the appropriate viewer.
9. Users can select a document's or folder's attributes for updating from the `CMBSearchResultsViewer` or `CMBFolderViewer`. Selecting a document generates a `CMBEditItemAttributesEvent`.
10. The `CMBItemAttributesEditor` bean listens for an `CMBEditItemAttributesEvent`. It displays the entity and attributes for the item. The user can then change the entity and attributes and then click **OK** to apply the changes.

Using beans in more than one window or dialog

You must provide additional code to pass an event from a bean in one window to a bean in another window. Typically, the fact that an event has been sent is usually the reason for displaying a window. The `EditAttributesDialog` window contains the `ItemAttributesEditor`. `SearchFrame` creates the window when a `CMBEditItemAttributesEvent` launches:

```
// Invoke a secondary dialog for edit attributes
searchResultsViewer.addEditItemAttributesListener(new
CMBEditItemAttributesListener() {
    public void onEditItemAttributes(CMBEditItemAttributesEvent event) {
        EditAttributesDialog editAttributesDialog = new
            EditAttributesDialog(SearchFrame.this,connection,event.getItem());
        editAttributesDialog.setVisible(true);
    }
});
```

The information that is normally passed to the `CMItemAttributesEditor` bean is passed as arguments to the constructor of the window instead. Within the constructor, the information is passed to the `CMItemAttributesEditor` bean by setting the following properties:

```
itemAttributesEditor.setConnection(connection);
```

```
itemAttributesEditor.setItem(item);
```

Chapter 11. Working with XML services (Java only)

The DB2 Content Manager Version 8 Release 3 connector provides APIs in `cmbxmlservice.jar` that can convert three types of system administration data into XML:

- Administration metadata objects, which can include users, user groups, resource manager configuration, workflow, ACL privileges, privilege sets, library server configuration, and the Information Integrator for Content search template and server configuration. This uses an XML schema called `cmadmin.xsd` (located in `IBMCMROOT/config/`) to represent the DB2 Content Manager Version 8 administration objects in an XML file.
- Data model metadata objects which define the structure for item types, component types, and Information Integrator for Content entities. These objects are stored as XSD files (*storage schemas*) that all refer to element or type definitions defined in the `cmdatamodel.xsd` file (located in `IBMCMROOT/config/`).
- Data instance objects, which can include items, attributes, and resource items (binary attachments). These objects are stored as XML files that conform to the storage schemas.

By representing objects as XML, you can import, store, update, retrieve, and export a wide variety of objects—such as documents or administrative data—between DB2 Content Manager servers without developing separate interfaces for each system.

This section explains the following tasks:

- “Understanding how XML services work with other DB2 Content Manager programming layers”
- “Importing and exporting DB2 Content Manager metadata using XML services” on page 432
- “Importing and exporting DB2 Content Manager data instance objects as XML” on page 454
- “Importing and exporting XML object dependencies” on page 457
- “Extracting content from different XML sources” on page 458
- “Mapping a user-defined schema to a storage schema with the XML schema mapping tool” on page 458
- “Programming runtime operations through the XML JavaBeans” on page 461

For a sample of the XML import and export functionality, see `TXMLExport.java`, `TXMLImport.java` and `TXMLList.java` in the `IBMCMROOT/samples/java/xml` directory.

Understanding how XML services work with other DB2 Content Manager programming layers

Certain programming layers require converted XML objects to work with the DB2 Content Manager connector. The layers include:

Web Services

DB2 Content Manager HTTP interface that accepts your XML messages (defined by `cmbmessages.xsd`) in a SOAP envelope to perform runtime operations such as import, export, search, create, update, retrieve, delete,

and document routing. The Web services automatically wrap and extract the XML messages in the SOAP message, and send them to the XML messaging JavaBean. For more information on Web services, see Chapter 12, “Working with the Web services,” on page 513

JavaBeans (XML)

Reusable Java classes based on the DB2 Content Manager connector XML APIs and the DB2 Information Integrator for Content JavaBeans. The XML JavaBeans perform runtime operations such as import, export, search, create, update, retrieve, delete, and document routing. In particular, the CMBXMLMessage bean parses all XML messages based on the cmbmessages.xsd schema. For more information on the XML JavaBeans see “Programming runtime operations through the XML JavaBeans” on page 461.

Schema mapping utility (XML)

XML conversion tool that can convert a user-defined schema into the storage schema that DB2 Content Manager supports. For more information on the schema conversion tool, see “Mapping a user-defined schema to a storage schema with the XML schema mapping tool” on page 458.

DB2 Content Manager connector (XML)

XML application programming interfaces that can import and export data model metadata objects, administrative metadata objects, and data instance objects.

Figure 34 illustrates how these Version 8 Release 3 XML layers relate to the DB2 Content Manager connector.

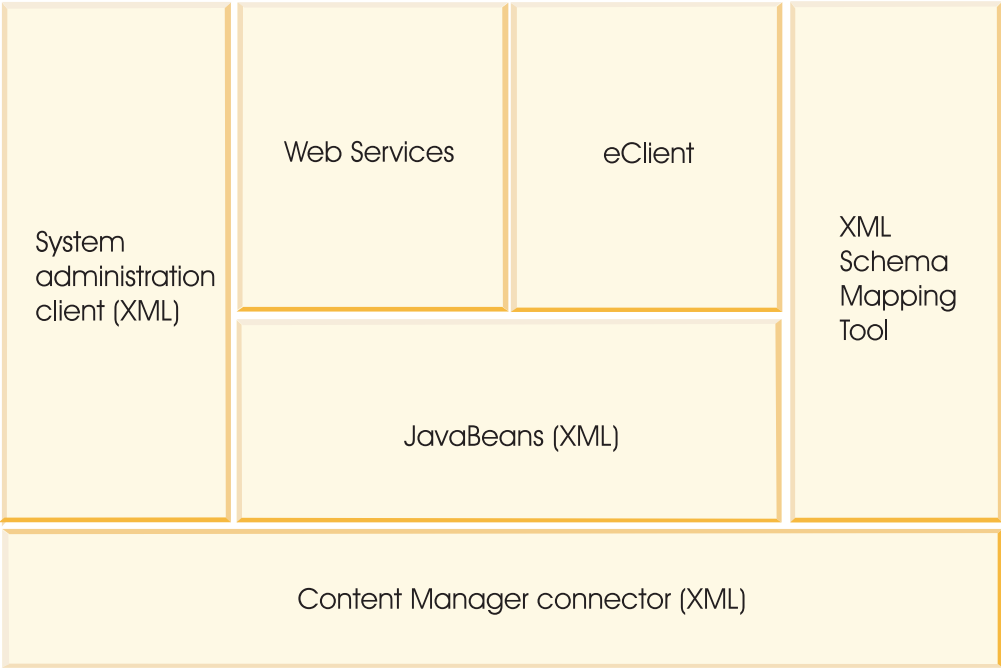


Figure 34. XML services programming layers

Figure 35 on page 431 depicts a real-world scenario for communicating with DB2 Content Manager through its XML interface:

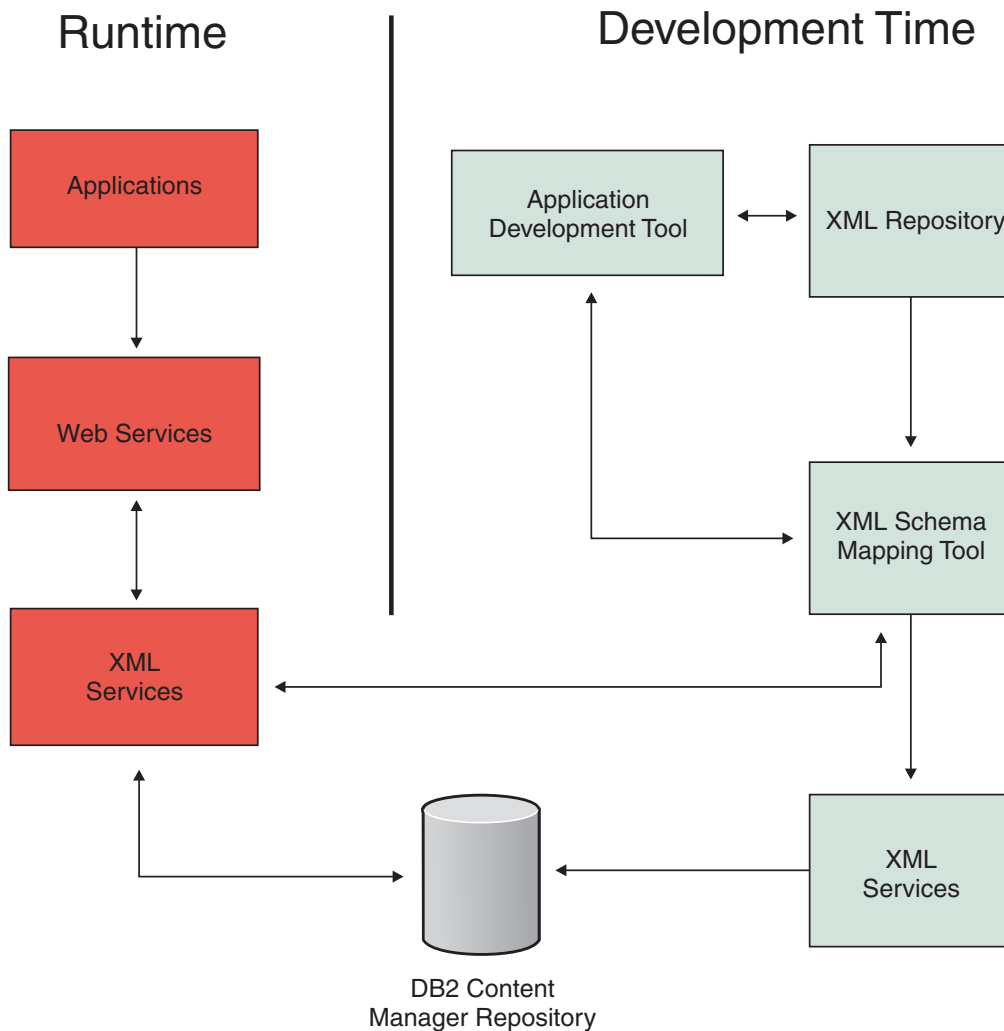


Figure 35. XML services communication flowchart

1. The XYZ insurance company defines schemas to hold its data, using application development tools like WebSphere Application Developer (WSAD) to create them. The company also chooses to store its data in DB2 Content Manager as items.
2. In an effort to migrate all data to DB2 Content Manager, the XYZ insurance company uses the DB2 Content Manager XML schema mapping tool to convert its schema to ones that DB2 Content Manager can parse, i.e., the storage schema.
3. The XYZ insurance company, through the XML schema mapping tool, creates the data model definition corresponding to the storage schema; then stores the schema mapping into DB2 Content Manager. The XML schema mapping tool internally invokes the ingest() API of the XML Services to create the data model definition.
4. The XYZ insurance company writes a custom application that uses Web service transactions to create, retrieve, update, delete, and route data in DB2 Content Manager. The Web services layer retrieves the schema mapping and transforms the data; then invokes the XML beans to perform the operation that manages data in the DB2 Content Manager.

Importing and exporting DB2 Content Manager metadata using XML services

You can import and export two types of metadata using the DB2 Content Manager APIs:

Administration objects

Contain users, user groups, resource manager configuration, workflow, ACL privileges, and the Information Integrator for Content search template and server configuration. This uses an XML schema called `cmadmin.xsd` (located in `IBMCMROOT/config/`) to define the XML files containing DB2 Content Manager Version 8 administration objects.

Data model objects

Define the structure of item types, component types, and Information Integrator for Content entities. These objects are stored as XSD files known as storage schemas. Each storage schema imports a common file named `cmdatamodel.xsd` (located in `IBMCMROOT/config/`).

By representing DB2 Content Manager metadata as intermediate files, you can program a number of scenarios:

- Customizing an application to administer and update data in DB2 Content Manager through the XML interface.
- Transferring metadata from one DB2 Content Manager system to another DB2 Content Manager system (taking into consideration any data conflicts).
- Transferring entities, search templates and server configuration from one Information Integrator for Content system to another Information Integrator for Content system (taking into consideration any data conflicts).

These scenarios become important in typical business situations such as:

- During the deployment of your content management application, transferring metadata from a test system to a production system.
- During the extension of an application or addition of a new DB2 Content Manager production system, transferring specific objects between development, test, and production systems. In this case, the existing data in the target system is updated.
- During troubleshooting of a production system, transferring specific objects from a production system to a test system in order to diagnose the problem.

The XML instance service class, `DKXMLSysAdminService`, contains three new Version 8 Release 3 methods for importing and exporting DB2 Content Manager metadata: `list()`, `ingest()`, and `extract()`. The latter two methods import and export storage schemas (XSD files) for data model objects, and XML files for administration objects (using the pre-defined `cmadmin.xsd` schema).

The `ingest()` and `extract()` methods support the following formats:

XML input formats

- `DKXMLStreamObjectDefs`: input stream
- `DKXMLDOMObjectDefs`: DOM

XML output formats

- `DKXMLDOMObjectDefs`: DOM

`DKXMLDOMObjectDefs` has a method to convert the DOM object into an output stream.

Additionally, DKXMLSysAdminService supports the following ingest() options:

DK_CM_XML_IMPORT_CONTINUE_ON_ERROR

If you OR this constant, then the ingest() method imports as much object definitions as possible. If you neither specify nor OR this constant, then the import process aborts on any error.

DK_CM_XML_IMPORT_CREATE_UPDATE

If you specify this constant, then the ingest() method replaces objects that already exist. If you do not specify this constant, then the import fails with an error for any object definitions that already exist.

The DKXMLDOMObjectDefs class provides two methods, getSysAdminDefs() and getDataModelDefs(), to retrieve the exported data model objects (in XML Schema format) and administrative objects (in XML format) separately. The DKXMLExportList class can specify which XML objects to export (for any objects that require other objects to work, see “Importing and exporting XML object dependencies” on page 457).

This section explains the following tasks:

- “Importing and exporting administration objects as XML”
- “Importing and exporting DB2 Content Manager data model objects as XML schema files (XSD)” on page 434
- “Unsupported XML types in the DB2 Content Manager storage schemas” on page 453

Importing and exporting administration objects as XML

The XML instance service class, DKXMLSysAdminService, contains two new Version 8 Release 3 methods for importing and exporting DB2 Content Manager metadata: ingest() and extract(). These methods import and export XML files for administration objects that conform to the cmadmin.xsd schema.

The DKXMLDOMObjectDefs class provides two methods, getSysAdminDefs() and getDataModelDefs(), to retrieve the exported data model objects (in XML Schema format) and administrative objects (in XML format) separately. The DKXMLExportList class can specify which XML objects to export.

The following DB2 Content Manager system administration objects (represented by constants in the com.ibm.mm.sdk.common.DKConstant class) can be converted to and from XML:

- Administrative domain
- Privilege definitions
- Privilege groups
- Privilege sets
- Users
- User Groups
- ACLs
- Library server configuration
- Library server language definition
- Link type
- MIME type
- Semantic type

- XDO class
- Resource manager objects
- Document Routing objects

The following DB2 Information Integrator for Content system administration objects can be converted to and from XML:

- Server definition
- Federated entities
- Search templates

The exported schema for a given data model object is semantically equivalent to the imported schema which creates it. That is, by exporting and importing an object from one system to another system, all the object properties should be the same to the original exported one. However, the exported schema document and the imported schema document might differ in syntax. This is because of the many different ways for XML Schema to represent the same information.

The following example creates a new user (Joshua) and a new group (XMLDev) in the DB2 Content Manager server:

XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<CMSSystemAdminDefinitions
  xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
  xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <user name="JOSHUA" adminDomainName="SuperDomain"
    defaultRM="RMDb"
    defaultSMSColl="CBR.CLLCT001" description="Regular user"
    passwordExpiration="0" userACL="PublicReadACL"
    userGrantPrivilegeSet="ClientUserReadOnly"
    userPrivilegeSet="AllPrivs">
  </user>
  <userGroup name="XMLDEV"/>
  </user>
  <userGroup adminDomainName="PublicDomain"
    description="XML Development" name="XMLDEV"/>
  <groupData groupName="XMLDEV">
    <user userName="JOSHUA"/>
  </groupData>
</CMSSystemAdminDefinitions>
```

Importing and exporting DB2 Content Manager data model objects as XML schema files (XSD)

The XML instance service class, DKXMLSysAdminService, contains two new Version 8 Release 3 methods for importing and exporting DB2 Content Manager metadata: ingest() and extract(). These methods import and export storage schemas (XSD files) for data model objects.

The DKXMLDOMObjectDefs class provides two methods, getSysAdminDefs() and getDataModelDefs(), to retrieve the exported data model objects (in XML Schema format) and administrative objects (in XML format) separately. The DKXMLExportList class can specify which XML objects to export.

Generally, the following rules apply when your storage schema is imported into DB2 Content Manager:

- A root element declaration (for example, an insurance policy) is mapped to an XYZ_InsPolicy item type.
`<xsd:element name="XYZ_InsPolicy">`
- A child element declaration (for example, a vehicle identification number) is mapped to an XYZ_VIN component type under the corresponding parent component type (in this example, the XYZ_InsPolicy root component type).
`<xsd:element maxOccurs="unbounded" minOccurs="0" name="XYZ_VIN">`
- An attribute inside of an element declaration is mapped to an attribute in the corresponding component (for example, a policy's ID number attribute maps to an XYZ_PolicyNum attribute in the policy item type).
`<xsd:attribute name="XYZ_PolicyNum">`

The following example shows a sample storage schema (XSD) snippet for the XYZ Insurance policy item type:

XML schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xsd:import namespace="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
    schemaLocation="cmdatamodel.xsd"/>
  <xsd:attribute name="XYZ_VIN"><xsd:annotation><xsd:appinfo>
    <cm:description value="Vehicle Identification Number (Content
      Manager Sample Attribute)" xsi:lang="ENU"/><cm:stringType
      value="OTHER"/></xsd:appinfo></xsd:annotation><xsd:simpleType>
    <xsd:restriction base="xsd:string"><xsd:length value="17"/>
    </xsd:restriction></xsd:simpleType></xsd:attribute>
  <xsd:attribute name="XYZ_InsrdLName">...</xsd:attribute>
  <xsd:attribute name="XYZ_InsrdFName">...</xsd:attribute>
  <xsd:attribute name="XYZ_ZIPCode">...</xsd:attribute>
  <xsd:attribute name="XYZ_City">...</xsd:attribute>
  <xsd:attribute name="XYZ_State">...</xsd:attribute>
  <xsd:attribute name="XYZ_Street">...</xsd:attribute>
  <xsd:attribute name="XYZ_PolicyNum">...</xsd:attribute>
  <xsd:element name="XYZ_InsPolicy"><xsd:annotation><xsd:appinfo>
    <cm:description value="Insurance Policy (Content Manager Sample
      Item Type)" xsi:lang="ENU"/><cm:ACL name="XYZInsurancePolicyACL"/>
    <cm:versionPolicy value="ALWAYS"/><cm:maximumVersions value="10"/>
    <cm:entityType value="DOCUMENT"/><cm:itemRetention unit="YEAR"
      value="0"/><cm:itemACLBinding flag="0"/><cm:itemEventFlag value=
      "0"/><cm:accessModule name="DUMMY" status="0" version="0"/><cm:
      previousAccessModule value="DUMMY"/></xsd:appinfo></xsd:annotation>
    <xsd:complexType><xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" ref="cm:properties"/>
      <xsd:element maxOccurs="1" minOccurs="0" ref="cm:links"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="XYZ_Insured">
        ...</xsd:element><xsd:element maxOccurs="unbounded" minOccurs="0"
        name="XYZ_VIN">...</xsd:element><xsd:element maxOccurs="unbounded"
        minOccurs="0" ref="ICMBASE">...</xsd:element><xsd:element
        maxOccurs="unbounded" minOccurs="0" ref="ICMBASETEXT">...
        </xsd:element><xsd:element maxOccurs="unbounded" minOccurs="0" ref=
        "ICMNOTELOG">...</xsd:element></xsd:sequence>...</xsd:complexType>
    </xsd:element>
    <xsd:element name="ICMBASETEXT">...</xsd:element>
    <xsd:element name="ICMNOTELOG">...</xsd:element>
  </xsd:schema>
```

The exported schema for a given data model object is semantically equivalent to the imported schema which creates it. That is, by exporting and importing an

object from one system to another system, all the object properties should be the same to the original exported one. However, the exported schema document and the imported schema document might differ in syntax. This is because of the many different ways for XML Schema to represent the same information.

DB2 Content Manager defines the storage schema using the following steps:

1. DB2 Content Manager attempts to map any available construct or feature in the XML schema to a DB2 Content Manager data model concept, as shown in Table 44.
2. If no DB2 Content Manager concept directly corresponds to the XML schema, then the concept is instead represented by a comment (also known as an XML annotation) as shown in Table 45 on page 443.
3. DB2 Content Manager instances (for example, item-level ACLs and semantic types) are represented as XML elements (in the DB2 Content Manager namespace) imported from the `cmdatamodel.xsd` file. This is shown in Table 45 on page 443.

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs

Object	Data type	Required	Corresponding storage schema construct	Comments
Attributes (global)			This represents an XML attribute in XML schema. For example: <code><attribute name=".." type=".." /></code> It is a global attribute declaration. Local attribute are described in the component type section.	
-Name	string	Yes	This represents an attribute name in the attribute declaration. For example: <code><attribute name="XX" type=".." /></code>	
-Type (possible type as follows)	short	Yes	This either maps to the built-in primitive and derived types, or with annotation.	The attribute's type is character, var char, short, long, etc.
--Character:			<p>This represents the derived string simple type definition with the <code>minLength</code> and <code>maxLength</code> constraints. For example: <code><simpleType name="char_100"> <restriction base="string"> <length value="100" /> </restriction> </simpleType></code></p> <p>Note that <code>minLength</code> and <code>maxLength</code> must be the same. There is one simple type definition per each attribute type instance with different length. The <code>simpleType</code> is tied to the attribute declaration in the export file. During import, the <code>simpleType</code> definition can be declared globally or inside the attribute declaration. The same definition can be re-used. Since the same attribute can exist in different item or components with different properties (such as length, nullable), it may not be possible to reference the global attribute declaration. As a result, the component has its own local attribute declaration. During the import, the XML Services API compares the definition and check whether there is any conflict.</p>	The length can be specified.
--alphanumeric				

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
--numeric				
--alphanumeric				
--Extended alphanumeric				
--other				
-Variable character			This represents the derived string simple type definition with the minLength and maxLength constraints. For example, <simpleType name="varchar_0,256"> <restriction base='string'> <minLength value="0"/> <maxLength value="256"/> </restriction> </simpleType>	The minimum and max length can be specified.
--alphabetic				
--numeric				
--alphanumeric				
--Extended alphanumeric				
--other				
-Short integer			This represents the short built-in derived type.	This is a specific min and max value.
-Long integer			This represents the integer built-in derived type.	This is a specific min and max value.
-Double			This represents the double built-in primitive type.	
-Decimal			This represents the decimal built-in primitive type.	This is a specific length and fixed places.
-Date			This represents the date built-in primitive type with facet value conversion (convert into the facet format which is understood by Content Manager and DB2).	
-Time			This represents the time built-in primitive type.	
-Timestamp			This represents the dateTime built-in primitive type with facet value conversion.	
-BLOB			A regular string simple type with a CM meta-attribute namespace. For example: <attribute name=".." type="base64Binary"> <annotation><appinfo> <CM:dataType value="BLOB"/> </appinfo> </annotation> </attribute>	The length can be specified.

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-CLOB			This maps to a regular string simple type with a CM meta-attribute namespace. For example: <code><attribute name="..." type="string"></code> <code><annotation><appinfo></code> <code><CM:datatype value="CLOB"/></code> <code></appinfo></annotation></code> <code></attribute></code>	The length can be specified.
-max	integer		The maxInclusive element in the derived simple type definition. For example: <code><simpleType name='integer_max100'</code> <code><restriction base='integer'</code> <code><maxInclusive value="100"/> </restriction></code> <code></simpleType></code> This will create a new type. DB2 Content Manager associates a property with the attribute.	This is only for short and integer.
-min	integer		The minInclusive element in the derived simple type definition. For example: <code><simpleType name='integer_min10' ></code> <code><restriction base='integer'</code> <code><minInclusive value="10" /></code> <code></restriction></code> <code></simpleType></code>	This is only for short and integer.
-length	integer		The length element in the derived simple type definition. For example: <code><simpleType name='char_256'</code> <code><restriction base="string"></code> <code><length value="256"/></code> <code></restriction></code> <code></simpleType></code>	
-scale/precision	integer		The fractionDigits and totalDigits elements in the derived simple type definition. For example: <code><simpleType name='decimal_8,3'</code> <code><restriction base='decimal'</code> <code><fractionDigits value="3"/></code> <code><totalDigits value="8"/></code> <code></restriction></code> <code></simpleType></code>	This is only for decimal.
Attribute groups (global)			The attributeGroup element in the XSD. For example: <code><xs:attributeGroup name="myAttrGrp"> <xs:attribute .../></code> <code>...</code> <code></xs:attributeGroup></code>	
-Name	string	Yes	The name attribute in the attributeGroup element. For example: <code><xs:attributeGroup name="myAttrGrp"> <xs:attribute .../></code> <code>...</code> <code></xs:attributeGroup></code>	

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-Attributes	array of string	yes	<p>The child elements of the attributeGroup element. For example, <xs:attributeGroup name="myAttrGrp"> <xs:attribute .../></p> <p>...</p> <p></xs:attributeGroup></p> <p>The attribute name will have the attribute group name as the prefix. This is consistent with how it is done in the CM. For the data instance, the attribute name is the combination of both the attribute group name and the attribute name. For example, <i>myAttrGrp.myStringAttr</i>.</p>	The is the array of attribute names.
Reference attribute			<p>This maps to a string attribute with a specific XML schema annotation.</p> <p>In the CM data model, reference attribute is mapped to an attribute group definition because it comes with few internal attributes; however, it does not match the data model. So instead of modeling it as an attributeGroup in XSD, you map it to a regular attribute.</p>	
-Name	string	yes	Maps to the attribute name.	
Item Type: definition			<p>A global element definition, with xs:schema element as the parent. For example:</p> <p><xs:schema></p> <p><xs:element name="elem1"></p> <p>....</p> <p></xs:element></p> <p></xs:schema></p> <p>It also has a similar XML Schema structure as the "Component Type Definition" because an item type is a component type based on the DB2 Content Manager data model.</p>	
-Name	string	yes	<p>The name attribute in the xs:element declaration. For example: <xs:schema></p> <p><xs:element name="elem1"></p> <p>....</p> <p></xs:element></p> <p></xs:schema></p>	
-Properties			Add an optional element definition right under the root element for describing any instance-level property. It will refer the element defined in the cmdatamodel.xsd schema file. For example: <element ref="cm:properties" minOccurs="0" maxOccurs="unbounded">	Use in the item instance to describe any instance-level properties associated with the item, such as semantic type, ACL, creation time, etc.
-Resource Object			Add an optional element definition right under the root element for describing any resource content associated with the item. It will refer the element defined in the cmdatamodel.xsd schema file. This element reference exists only if the element definition is describing a resource item type or a part item type. <element ref="cm:resourceObj" minOccurs="0" maxOccurs="unbounded">	Use in the item instance to describe any resource content associated with the item.

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-Embedded Link object			Add an optional element definition right under the root element for describing any inbound and outbound links in the instance level. It will refer the element defined in the cmdatamodel.xsd schema file. This element reference exists only if the element definition is describing an item type, a resource item type or a document item type. <element ref="cm:links" minOccurs="0" maxOccurs="unbounded">	Use in the item instance to describe all the inbound and outbound links.
Item type classification	short			Represents the item, resource item, document, or document part. Use the ItemTypeRelation DefICM to represent relationships to the part.
<ul style="list-style-type: none"> • Non-resource item • Resource item • Document item type 				
-Part type name	string,int		Add an optional element under the root element to reference the defined PART element declaration. For example: <element ref="cm:ICMBASE" minOccurs="0" maxOccurs="unbounded"> The part element definition will have the <cm:referencedOnly> annotation element if the part item type is not part of the export list.	
-Text searchable /option				
-ACL name	string,int			
-RM name	string,short			
-SMS coll name	string,short			
-New version policy	short			This never creates, always creates or the user chooses
4. Document part type			This represents a root element declaration (an item type). Such root element can only be referred by a document type. No XML instance can be based on this document type.	
Component Type:			This maps to an element declaration in XML schema. If the definition exists in a standalone fashion (it means it does not associate with any itemType), it maps to a root element declaration in the XML schema with a cm:entityType element with value="component". Mostly non-root element--children of another element. This would be root element if there is no item type definition it corresponds to. This accepts only "sequence" , but not "all", "choice" and nested particle. The DB2 Content Manager data model does not track the instance order.	

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-Attribute groups			This maps to XSD attribute group definition with "ref=" to the global attribute group definition. The global attribute group definition will have the name: <component type name>.<attribute group name>. The global attribute group definition has the <cm:referencedOnly> annotation element to distinguish it from the exported global attribute group.	
-Reference attributes			This maps to XSD attribute use. It has the same local definition as the global reference attribute definition.	
-Attributes			This maps to XSD attribute use. The local attribute has its own type definition and annotation. It will use all the type definition and annotations described in the global attribute, which applies to local.	
-Is required	boolean		This represents the used attribute name in the attribute declaration. For example: <xsd:attribute name=".." use="required" .../> Here is how you map the value of nullable attribute to the "use" and "default" attribute. "nullable --> use="optional" "non-nullable w/o default --> use="required" "non-nullable w/ default --> use="optional" default="X"	
-Max value	integer		This defines a new simpleType with a new max value, such as the maxInclusive element in the derived simple type definition. For example: <simpleType name='integer_max100'> <restriction base='integer'> <maxInclusive value="100"/> </restriction> </simpleType>	
-Min value	integer		This represents the minInclusive element in the derived simple type definition. For example: <simpleType name='integer_min10'> <restriction base='integer'> <minInclusive value="10"/> </restriction> </simpleType>	
-Default value	string		This represents the default attribute in the attribute declaration. For example: <xsd:attribute name=".." default="10" ... />	
Sub-components / child components			This maps to a child element declaration (using sequence) with the min and max occurrence of a child component mapped to the minOccur and maxOccur of the element declaration.	

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
Item Type View:			This maps to a root element definition. The item type view definition shares the same definition as item type definition with only the annotations marked with scope = "VIEW". It also inherits the definition of the component type view because an item type view itself is a component type view.	This performs projection, not selection.
Component Type View:			This maps to a child element definition of the item type view definition. It will have the same hierarchy structure as component type definition. For example it contains attributes, attribute groups and sub-components.	This performs projection, not selection.
-Attributes			Use the same representation as the regular attribute definition in component type.	
Federated entity			This maps to a root element definition. Since this is a federated entity, it cannot contain any sub-element. Such entity definitions can only be created through the Federated connector.	
-name	string	yes	The name attribute in the xs:element declaration. For example: <xs:schema> <xs:element name="elem1"> <xs:annotation><xs:appinfo> <xs:entityType value="federated" /> </xs:appinfo></xs:annotation> </xs:element> </xs:schema>	
-Text searchable	boolean			
-Enabled creating the native federated folder	boolean			
-Attribute:			This maps to an XSD attribute declaration within the element.	
--name	string		The name attribute in the xs:attribute declaration. For example: <xs:schema> <xs:element name="elem1"> <xs:annotation><xs:appinfo> <xs:entityType value="federated" /> </xs:appinfo></xs:annotation> <xs:complexType> <xs:attribute name="attr1"> </xs:element> </xs:schema>	

Table 44. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
--datatype and other attribute information, such as length, precision, scale, max, min, nullable, queryable, writeable				Use the same definition as in the attribute category, but some representation will not be there (such as the text-searchable information).
-Fed entity name	string			
-Fed attr name	string			
-Native server name	string			
-Native server type	string			
-Native entity name	string			

For Table 45, the **Scope of the annotation** can have the following definitions:

GLOBAL

The annotation applies if the property belongs to a global definition or declaration.

LOCAL

The annotation applies if the property belongs to a local declaration.

VIEW The annotation applies only if the property belongs to a declaration which is part of the ITEM TYPE VIEW.

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema

Object	Information specified in annotation	Scope of the annotation	Instance-level information
Attributes (global)			
-Type (possible type as follows)			
--Character:			
--alphabetic	You use the element cm:stringType with a value attribute in enumeration type {ALPHA, NUMERIC, ALPHANUMERIC, ALPHANUMERIC_EXT, OTHER}, to represent the value. Putting the pattern element will be optional. In this case, <cm:stringType value="ALPHA" />	GLOBAL, LOCAL, VIEW	
--numeric	<cm:stringType value="NUMERIC" />	GLOBAL, LOCAL, VIEW	
--alphanumeric	<cm:stringType value="ALPHANUMERIC" />	GLOBAL, LOCAL, VIEW	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
--Extended alphanumeric	<cm:stringType value="ALPHANUMERIC_EXT" />	GLOBAL, LOCAL, VIEW	
--other	<cm:stringType value="OTHER" />	GLOBAL, LOCAL, VIEW	
-Variable character			
--alphabetic	Same as in the Character case <cm:stringType value="ALPHA" />	GLOBAL, LOCAL, VIEW	
--numeric	<cm:stringType value="NUMERIC" />	GLOBAL, LOCAL, VIEW	
--alphanumeric	<cm:stringType value="ALPHANUMERIC" />	GLOBAL, LOCAL, VIEW	
--Extended alphanumeric	<cm:stringType value="ALPHANUMERIC_EXT" />	GLOBAL, LOCAL, VIEW	
--other	<cm:stringType value="OTHER" />	GLOBAL, LOCAL, VIEW	
-Short integer			
-Long integer			
-Double			
-Decimal			
-Date			
-Time			
-Timestamp			
-BLOB	<CM:dataType value="BLOB" />	GLOBAL, LOCAL, VIEW	
-CLOB	<CM:dataType value="CLOB" />	GLOBAL, LOCAL, VIEW	
-max			
-min			
-length			
-scale/precision			
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
Attribute groups (global)			
-Name			

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-Attributes			
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
Reference attribute	Indicated by an annotation element, cm:referenceAttribute with the following attributes:	GLOBAL, LOCAL, VIEW	
-Name			
-Reference delete rule	{cm:referenceAttribute} cm:deleteRule=<string> in enumeration type {NO_ACTION, SET_NULL, CASCADE, RESTRICT.	GLOBAL, LOCAL, VIEW	
-Reference sequence number	{cm:referenceAttribute} cm:sequenceNumber=<short>	GLOBAL, LOCAL, VIEW	
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
Item Type: definition			
-Name			
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
-New version policy	cm:versionPolicy element with a value attribute in enumeration type {NEVER, ALWAYS, BY_APPLICATION}.	GLOBAL	
-Maximum total versions	cm:maximumVersions element with a value attribute.	GLOBAL	
-Properties			The cmdatamodel.xsd file contains the definition of the properties.
-Resource Object			The cmdatamodel.xsd3 file contains the definition of the resourceObj.

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-Embedded Link object			<p>The cmdatamodel.xsd file contains the definition of the links, which is <element name="links"></p> <pre> <complexType> <sequence> <element name="outbound" minOccurs="0" maxOccurs="unbounded"> <complexType> <attribute name="toitem" type="string"/> <attribute name="linktype" type="string"/> <attribute name="linkitem" type="string"/> </complexType> </element> <element name="inbound" minOccurs="0" maxOccurs="unbounded"> <complexType> <attribute name="fromitem" type="string"/> <attribute name="linktype" type="string"/> <attribute name="linkinfoitem" type="string"/> </complexType> </element> </complexType> </element> </pre> <p>Note that this will be part of the pre-defined DB2 Content Manager schema file, cmdatamodel.xsd, to be imported.</p>
Item type classification	Implied by the following elements:		
1. Non-resource item	If cm:entityType does not exist or cm:entityType="ITEM".	GLOBAL, VIEW	
2. Resource item (includes the following info)	A cm:entityType element with the value="RESOURCEITEM" to indicate this is a resource item type. Also there should be a cm:resourceItemInfo element with the following attributes/elements.	GLOBAL, VIEW	
-XDO class name	{cm:resourceItemInfo} a cm:name attribute in the cm:XDOClass child element of cm:resourceItemInfo element.	GLOBAL	
-Text Searchable	{cm:resourceItemInfo} a cm:textSearchable element, with a value attribute in boolean type.	GLOBAL	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-Text Index Info	{cm:resourceItemInfo} same set of attributes in the cm:textIndexInfo element as the text information in the attribute declaration. But in this case, it will be in the item type level and the element will be a child element of cm:resourceItemInfo.	GLOBAL	
-RM name	{cm:resourceItemInfo} cm:RM child element of cm:resourceItemInfo element with name and attributes.	GLOBAL	
-SMS coll name	{cm:resourceItemInfo} cm:SMSCollection child element of cm:resourceItemInfo element with name attributes.	GLOBAL	
-Prefetch coll name	{cm:resourceItemInfo} cm:prefetchCollection element with name and attributes.	GLOBAL	
-an embedded binary object (used in the instance level)			<p>The cmdatamodel.xsd contains the definition of the resourceObject, which is</p> <pre><element name="cm:resourceObject"> <complexType> <choice> <element name="content"> <complexType> <attribute name="value" type="base64Binary" /> </complexType> </element> <element name="url"> <complexType> <attribute name="value" type="anyURI" /> </complexType> </element> </complexType> </element></pre> <p>Note that the type would be either "base64Binary" or "hexBinary". This will be part of the pre-defined DB2 Content Manager schema file, cmdatamodel.xsd, to be imported.</p>
3. Document item type (includes the following info)	A cm:entityType element with the value="DOCUMENT" to indicate this element declaration is a document type.	GLOBAL, VIEW	There are several pre-defined ICMPARTs (ICMANNOTATION, ICMBASE, ICMBASETEXT, ICMBASESTREAM, ICMNOTELOG). They will be handled or represented similar to the regular user part, which will be part of the schema file.
-Part type name			
-Text searchable /option	Same case as in the resource item type definition, having the same cm:textsearchable and cm:textIndexInfo elements under the cm:resourceItemInfo element.	GLOBAL	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-ACL name	cm:ACL element with name and attributes under the referenced element. For example: <element ref="cm:ICMBASE" ...> <xsd:annotation> <xsd:appinfo> <cm:ACL name="..." />	GLOBAL	
-RM name	cm:RM element with name and attributes under the reference element name.	GLOBAL	
-SMS coll name	cm:SMSCollection element with name and attributes under the reference element name.	GLOBAL	
-New version policy	cm:versionPolicy element name with a value attribute in enumeration type {NEVER, ALWAYS, BY_APPLICATION}.	GLOBAL	
4. Document part type	A cm:entityType element with the value="PART" to indicate this element is a document part type.	GLOBAL, VIEW	If a CM pre-defined PART type, it will be in the cmdatamodel.xsd file.
-XDO class name	cm:XDOClass element with a value attribute.	GLOBAL	
-Text searchable /option	Same case as in the resource item type definition, having the same cm:textsearchable and cm:textIndexInfo elements under the cm:resourceItemInfo element.	GLOBAL	
Item retention period	cm:itemRetention element with value and unit attributes. The unit attribute is in enumeration type {YEAR, MONTH, WEEK, DAY}.	GLOBAL	
Start item on process	cm:startProcess element with a name attribute.	GLOBAL	
Default Priority	cm:defaultPriority element with a value attribute.	GLOBAL	
Description	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
Item type level ACL Name	cm:ACL element with name and attributes.	GLOBAL	
Item Level ACL Binding Flag	cm:itemACLBinding element with a flag attribute.	GLOBAL	
Auto Linking	cm:autoLinkEnable element with the following child elements.	GLOBAL	
-Auto Linking Rules	a cm:autoLinkAttributes element with the following attributes:		
-Current item type	{cm:autoLinkAttributes} sourceItemType=<string>	GLOBAL	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-Item type to be linked	{cm:autoLinkAttributes} cm:targetItemType=<string>	GLOBAL	
-Auto linking attribute	{cm:autoLinkAttributes} cm:attributeName=<string>	GLOBAL	
-Auto linking attribute	{cm:autoLinkAttributes} cm:attributeGroupName=<string>	GLOBAL	
-Link type	{cm:autoLinkAttributes} cm:autoLinkType=<string>	GLOBAL	
-Auto Linking SMS logging	A cm:autoLinkingSMSRule element with a value attribute.	GLOBAL	
-Item type events to log	cm:itemEventFlag element with value attribute.	GLOBAL	
Foreign keys	A cm:foreignKey element with the following attributes:	GLOBAL	
-Constraint name	{cm:foreignKey} cm:constraintName=<string>	GLOBAL	
-Update rule	{cm:foreignKey} cm:updateRule={RESTRICT, NO_ACTION}	GLOBAL	
-Delete rule	{cm:foreignKey} cm:deleteRule={RESTRICT, CASCADE, NO_ACTION, SET_NULL}	GLOBAL	
-Source component	{cm:foreignKey} cm:sourceComponent=<string>	GLOBAL	
-Target item type	{cm:foreignKey} cm:targetItemType=<string>	GLOBAL	
-Target external table	{cm:foreignKey} cm:targetTable=<string>	GLOBAL	
-Attribute Pairs	A cm:attributePair element under the cm:foreignKey element with the following attributes:	GLOBAL	
-Source attribute group	{cm:attributePair} cm:sourceAttributeGroup=<string>	GLOBAL	
-Source attribute	{cm:attributePair} cm:sourceAttribute=<string>	GLOBAL	
-Target attribute group	{cm:attributePair} cm:targetAttributeGroup=<string>	GLOBAL	
-Target attribute	{cm:attributePair} cm:targetAttribute=<string>	GLOBAL	
-External table column name	{cm:attributePair} cm:targetTableColumn=<string>	GLOBAL	
User exits	cm:userExit element with the following attributes:	GLOBAL, VIEW	
-Exit name	{cm:userExit} cm:name=<string>	GLOBAL, VIEW	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-Function name	{cm:userExit} cm:functionName=<string>	GLOBAL, VIEW	
-DLL name	{cm:userExit} cm:DLLName=<string>	GLOBAL, VIEW	
Access Module	A cm:accessModule element with the following attributes:	GLOBAL	
-name	{cm:accessModule} cm:name=<string>	GLOBAL	
-result	{cm:accessModule} cm:result=<integer>	GLOBAL	
-status	{cm:accessModule} cm:status=<short>	GLOBAL	
-version	{cm:accessModule} cm:version=<short>	GLOBAL	
Previous Access Module	A cm:previousAccessModule element with a version attribute.	GLOBAL	
Component Type:			
-Attribute groups			
-Reference attributes			
-Attributes			
-Text Searchable	A cm:textSearchable element, with a value attribute in boolean type.	GLOBAL, LOCAL, VIEW	
-Text Index Information	A cm:textIndexInfo element, with the following attributes:.	GLOBAL, LOCAL	
-Commit count	{cm:textIndexInfo } cm:commitCount=<integer>	GLOBAL, LOCAL	
-Format	{cm:textIndexInfo } cm:format=<integer>	GLOBAL, LOCAL	
-Index CCSID	{cm:textIndexInfo } cm:CCSID=<integer>	GLOBAL, LOCAL	
-Index Directory	{cm:textIndexInfo } cm:directory=<string>	GLOBAL, LOCAL	
-Index Language Code	{cm:textIndexInfo } cm:langCode=<string>	GLOBAL, LOCAL	
-Minimum changes	{cm:textIndexInfo } cm:minChanges=<integer>	GLOBAL, LOCAL	
-Model File	{cm:textIndexInfo } cm:modelFile=<string>	GLOBAL, LOCAL	
-Model Name	{cm:textIndexInfo } cm:modelName=<string>	GLOBAL, LOCAL	
-Model CCSID	{cm:textIndexInfo } cm:modelCCSID=<integer>	GLOBAL, LOCAL	
-UDF Name	{cm:textIndexInfo } cm:UDFName=<string>	GLOBAL, LOCAL	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-UDF Schema	{cm:textIndexInfo } cm:UDFSchema=<integer>	GLOBAL, LOCAL	
-Update Frequency	{cm:textIndexInfo } cm:updateFrequency=<string>	GLOBAL, LOCAL	
-Update Frequency Unit	{cm:textIndexInfo } cm:updateFrequencyUnit= <{MINUTE, HOUR}>	GLOBAL, LOCAL	
-Working Directory	{cm:textIndexInfo } cm:workingDir=<string>	GLOBAL, LOCAL	
-Is representing item	cm:representative element with a value attribute in boolean type.	LOCAL	
-Is unique	cm:unique element with a value attribute in boolean type.	LOCAL	
-Is required			
-Max value			
-Min value			
-Default value			
-Resource Manager attribute	cm:isResourceManagerAttr element with a value attribute in boolean type.	LOCAL	
-Database indexes	A cm:databaseIndexInfo element, with the following attributes:	GLOBAL, LOCAL	
--Name	{cm:databaseIndexInfo } cm:name=<string>	GLOBAL, LOCAL	
--Unique	{cm:databaseIndexInfo } cm:unique=<boolean>	GLOBAL, LOCAL	
--Index Schema	{cm:databaseIndexInfo } cm:indexSchema=<string>	GLOBAL, LOCAL	
--Attributes	A subelement, cm:indexedAttribute element under the same cm:databaseIndexInfo element with the following attributes:	GLOBAL, LOCAL	
---Name	{cm:indexAttribute } cm:name=<string>	GLOBAL, LOCAL	
---Index Order	{cm:indexAttribute } cm:indexOrder={ASCENDING, DESCENDING}	GLOBAL, LOCAL	
Delete rule	A cm:deleteRule element with a value attribute {RESTRICT, CASCADE}.	LOCAL	
Sub-components/ child components			
Item Type View:	A cm:entityType element with the value="ITEM" and a cm:entityView element with cm:baseEntityType attribute to indicate which item type this refers to.	VIEW	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
Component Type			
View:			
-Attributes	cm:representative element with a value attribute in boolean type	VIEW	
--readable	cm:readable element with a value attribute in boolean type. No such concept exists in the XML schema. Only has "fixed" in term of the value, or "prohibited" in term of attribute name definition.	VIEW	
--writable	cm:writeable element with a value attribute in boolean type	VIEW	
--queryable	cm:queryable element with a value attribute in boolean type	VIEW	
--excludeRow	cm:excludeRow element with a value attribute in boolean type	VIEW	
--View compare value	cm:viewCompareValue element with a value attribute	VIEW	
--View operator	cm:viewOperator element with a value attribute in enumeration type {OPCODE_EQ}	VIEW	
--View sequenceNumber	cm:viewSequenceNo element with a value attribute	VIEW	
Federated entity	A cm:entityType element with the value="federated"		
-name			
-description			
-Text searchable			
-Enabled creating the native federated folder			
-Attribute:			
--name			
--description	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to)	GLOBAL	
--datatype and other attribute information, such as length, precision, scale, max, min, nullable, queryable, writeable			
Schema mapping:	One or more cm:schemaMapping elements with the following attributes:	GLOBAL	

Table 45. How DB2 Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-Fed entity name	cm:fedEntityName=<string>	GLOBAL	
-Fed attr name	cm:fedAttrName=<string>	GLOBAL	
-Native server name	cm:serverName=<string>	GLOBAL	
-Native server type	cm:serverType=<string>	GLOBAL	
-Native entity name	cm:nativeEntityName=<string>	GLOBAL	
-Native attr name	cm:nativeAttrName=<string>	GLOBAL	

Unsupported XML types in the DB2 Content Manager storage schemas

The DB2 Content Manager storage schemas do not support the following primitive datatypes:

- string (has to be associated with either length properties)
- boolean
- duration
- gYearMonth
- gYear
- gMonthDay
- gDay
- gMonth
- hexBinary (only support base64Binary)
- QName
- NOTATION

The DB2 Content Manager storage schemas do not support the following derived datatypes:

- normalizedString
- token
- language
- NMTOKEN
- NMTOKENS
- Name
- NCName
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- nonPositiveInteger
- negativeInteger
- long
- byte

- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger

The following constraints also apply:

- The minLength and maxLength attribute values (if specified) must be the same, which describes the DK_CM_CHAR data type.
- An element name (which maps to component name) cannot be used in different symbol space.
- Any attribute with the same name must have the same basic type. **Exception:** Some properties of the attributes with the same name can be different, such as maxInclusive and minInclusive.
- The xsi:type and xsi:nil attributes are not supported in the instance document.
- No recursive type definition is allowed. For example, the following definition is not allowed:

```
<xs:element name="Section">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Section" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="title" use="required"/>
    <xs:attribute ref="content" use="required"/>
  </xs:complexType>
</xs:element>
```

Importing and exporting DB2 Content Manager data instance objects as XML

The XML instance service class, DKXMLDataInstanceService, contains two new Version 8 Release 3 methods for importing and exporting XML items: ingest() and extract(). These methods take the XML file which conforms to the storage schema (described in “Importing and exporting DB2 Content Manager data model objects as XML schema files (XSD)” on page 434, which can be exported through the extract API or the system administration client, on any item type) to structure the data instance objects (such as items and documents with parts). Their input and return parameters work similarly to the old Version 8.2 methods, toXML() and fromXML().

The ingest() and extract() methods support the following formats:

XML item input formats

- DKXMLDOMItem: document object model (DOM):
- DKXMLStreamItem: input stream (processed using SAX)
- DKXMLStringItem: string

XML item output formats

- DKXMLDOMItem: DOM (default)
- DKXMLStringItem: string

DKXMLDOMItem features a method that can convert an XML item from DOM format to Input stream format.

The following example shows a sample item instance that conforms to the XYZ Insurance policy storage schema:

XML item instance

```
<Item><ItemXML>
<XYZ_InsPolicy XYZ_Street="532 Camino Viejo"
  XYZ_City="Marina" XYZ_State="CA" XYZ_ZIPCode="90546"
  XYZ_PolicyNum="57904965371" xmlns="">
  <XYZ_Insured XYZ_InsrdFName="Edward" XYZ_InsrdLName="Smith" />
  <XYZ_Insured XYZ_InsrdFName="Jennifer" XYZ_InsrdLName="Smith" />
  <XYZ_VIN XYZ_VIN="ICLA44P5KL9876543" />
  <ICMBASE><resourceObject MIMEType="image/tiff"
    xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"><label
      name="policyForm" /></resourceObject></ICMBASE>
</XYZ_InsPolicy>
</ItemXML></Item>
```

This section explains the following tasks:

- “Exporting DB2 Content Manager DDO items as XML items”
- “Importing XML items as DB2 Content Manager DDO items” on page 456

Exporting DB2 Content Manager DDO items as XML items

The extract() method in DKXMLDataInstanceService converts a DDO (including all child component DDOs, links, and references) into a DKXMLItem object. This DKXMLItem object contains the following data:

- XML document that represents the item version, including all child components.
- Properties, including system attributes, resource attributes, and links information.
- Any binary resource part content. **Recommendation:** Pass in the DKConstant.DK_CM_XML_EMBED_UNIQUE_IDENTIFIER to TRUE in order to include the resource content’s part number.

The extract() method accepts the DDO (and various options) as the input parameters. The various options include:

- Which XML format to export the item and its properties:
 - DKXMLDOMItem: DKConstant.DK_CM_XML_DOM_FORMAT (this is the default)
 - DKXMLStreamItem: DKConstant.DK_CM_XML_RESOURCE_STREAM_FORMAT
 - DKXMLStringItem: DKConstant.DK_CM_XML_DOM_FORMAT
- Which output format to export the resource content as (URL or input stream). URL is the default. If you select input stream, then the system generates unique labels to identify each resource. These labels can be found in the resource properties.
- Whether to include the PID and part number with the XML item.
- Whether to export the item’s properties as a separate XML document. You would use this option to exclude all proprietary Content Manager information from the XML item.

The following example inputs a ddo item and returns it as an xmlobj XML document (in DOM format with the PID embedded in it); returns system and resource properties in a separate file; returns resource content as an input stream.

Java

```
//create an instance of DKXMLDataInstanceService
DKXMLDataInstanceService instService = new DKXMLDataInstanceService(dsICM);
DKXMLDOMItem xmlObj =
(DKXMLDOMItem) instService.extract(ddo,DKConstant.DK_CM_XML_DOM_FORMAT +
DKConstant.DK_CM_XML_SYSTEM_PROPERTY_REFERENCE +
DKConstant.DK_CM_XML_RESOURCE_PROPERTY_REFERENCE +
DKConstant.DK_CM_XML_RESOURCE_STREAM_FORMAT +
DKConstant.DK_CM_XML_EMBED_UNIQUE_IDENTIFIER);
//get the XML document representing item version
Document xmlDocument = xmlObj.getXMLItem();
//get the XML document with properties
Document propertyDocument = xmlObj.getItemProperties();
//get content labels
Set resLabels = xmlObj.getContentLabels();
//create an iterator
Iterator iter = resLabels.iterator();
//iterate over the set to get labels and resource contents
while (iter.hasNext()) {
    //get the label from the iterator
    String label = (String)iter.next();
    // get resource content as input stream from xml object
    BufferedInputStream inStream = new
    BufferedInputStream(xmlObj.getContentAsStream(label));
}
```

Importing XML items as DB2 Content Manager DDO items

The ingest() method in DKXMLDataInstanceService converts a DKXMLItem object into a DDO on the Content Manager server. 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 DB2 Content Manager. The new DDO belongs to a DB2 Content Manager Version 8 item type or an earlier Content Manager index class and can only be stored in DB2 Content Manager. Importing an 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.

As you import XML content, keep the following facts in mind:

- You can only import into DB2 Content Manager or earlier DB2 Content Manager.
- XML files containing content for import must conform to the storage schema of the corresponding item type, which you can export through the API described in “Importing and exporting DB2 Content Manager data instance objects as XML” on page 454 or the system administration client.
- XML import and XML export are supported only by the Java APIs.

The input() method accepts the following input parameters:

- XML document in a DKXMLDOMItem, DKXMLStreamItem, or DKXMLStringItem. If you input a DKXMLStreamItem, then a SAX handler converts the input stream to a DDO object (not DOM).
- A pre-existing DDO to populate the XML data with (optional).

- Resource content as a DKXMLItem object in input stream format. Using the DKXMLItem.setContentAsStream() method, you can create unique labels for the resource properties for ingest() to interpret.
- Properties such as system attributes, resource attributes, and links information. You can either embed them in the original XML document, or import them as a separate XML document from the original XML document which describes the item. You can either provide this document through the setItemProperties() method or in the constructor.

The following example inputs both an XML item XMLFile and its properties (both system and resource in a separate file XMLProperties) as input streams; and returns a DB2 Content Manager ddo.

Java

```
//create file stream for XML document representing item version
FileInputStream xmlDocument = new FileInputStream(XMLFile);
//create file stream for XML document representing properties
//properties include system properties, resource properties
FileInputStream properties = new FileInputStream(XMLProperties);
//Create an instance of DKXMLStreamItem
DKXMLStreamItem xmlItem = new DKXMLStreamItem(XMLFile, XMLProperties);
//set value for resource content label
String contentLabel = "AAA";
//Set resource content into xmlItem
xmlItem.setContentAsStream(contentLabel, contentStream);
//create an instance of DKXMLDataInstanceService
DKXMLDataInstanceService instService = new DKXMLDataInstanceService(dsICM);
//call ingest on instance service
DKDDO ddo = (DKDDO) instService.ingest(xmlItem, options);
ddo.add()
```

Importing and exporting XML object dependencies

Scenarios can occur where data model and administration objects require the existence of other definitions (*dependency objects*) in the server. For example, a user must be defined before you can define a user group for it.

By default, the extract() method only exports the object and no dependency objects. In order to prevent problems from missing dependencies, you can specify one of the following options for exporting objects to XML:

DK_CM_XML_EXPORT_PREREQUISITE

Exports all dependency objects with the object.

DK_CM_XML_EXPORT_DM_ONLY_PREREQUISITE

Exports all data model dependency objects only.

DK_CM_XML_EXPORT_SA_ONLY_PREREQUISITE

Exports all administration dependency objects only (including authorization, authentication, and library server configuration).

DK_CM_XML_EXPORT_RM_ONLY_PREREQUISITE

Exports all resource manager configuration (in the library server side) dependency objects only.

DK_CM_XML_EXPORT_DR_ONLY_PREREQUISITE

Exports all document routing dependency objects only.

During an import, if the dependency objects do not exist in the target system, an exception is logged or the process is aborted (depending on the error handling option set).

Extracting content from different XML sources

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

Here are examples of each XML source:

XML from a file

Java

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

XML from a buffer

Java

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

XML from a Web address (URL)

Java

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

Mapping a user-defined schema to a storage schema with the XML schema mapping tool

DB2 Content Manager provides both a graphical interface and APIs to convert a user-defined schema into a storage schema that can be imported into the system. The tool can also generate an XSLT query script which can be saved as part of a mapping in a repository. Using this script, you can program an application that automatically converts XML documents from the user-defined schema to the storage schema. For details in the graphic interface, see the "mapping and importing XML schemas" topic in the system administration guide.

The XML schema mapping tool supports the following scenarios when developing your schema mapping:

Creating schema mappings with a brand-new storage schema

You can convert your user-defined schema to a brand-new storage schema,

and you can modify both the storage schema and mappings. You can then create a new item type from the storage schema, assign a mapping name, and save the mapping in a repository.

Creating schema mappings with a pre-existing storage schema

You can convert your user-defined schema to a previously created storage schema by manually mapping the user schema to the storage schema. You can then invoke the tool function that will generate a new XSLT query script. You can then assign a mapping name and save the mapping in a repository.

Revise existing schema mappings

You can re-open a previously created mapping (using the mapping name), and modify both it and the storage schema. You can then save the modified storage schema, user-defined schema, and new XSLT query script back to the repository.

As a first step in using the APIs, you would use methods in the `DKSchemaConverter` class to convert your XML schema. The methods perform the following tasks:

convert()

Converts the user-defined schema to a storage schema and optionally saves the mapping as an XSLT query script in a repository.

getStorageSchema()

Retrieves the converted storage schema.

getXSLTQuery()

Retrieves the XSLT query script that can automatically convert XML documents from the user-defined schema to the storage schema.

Java

```
import com.ibm.mm.sdk.common.DKException;
import com.ibm.mm.sdk.cs.DKDatastoreICM;
import com.ibm.mm.sdk.xml.schema.DKDocumentConverter;
import com.ibm.mm.sdk.xml.schema.DKMapperException;
...
DKDatastoreICM cmDatastore = new DKDatastoreICM();
cmDatastore.connect(cmDatabase, cmUser, cmPassword, "");
System.out.println("Connected.");
File inputSchema = new File ( inputUserSchema );
DKSchemaConverter converter = new DKSchemaConverter( cmDatastore );
if (mapName == null) {
    if (converter.convert( inputSchema.toURL(), rootElementName)==false)
    {
        System.err.println("dkConvert returned null.");
    }
} else {
    if (converter.convert( inputSchema.toURL(), rootElementName,
mapName ) == false)
    {
        System.err.println("dkConvert returned null.");
    }
}

System.out.println("STORAGE SCHEMA:");
System.out.println( converter.getStorageSchema() );
System.out.println("XSLT Scripts");
String scripts[] = converter.getXSLTQuery();
System.out.println( scripts[0] );
System.out.println("-----");
System.out.println( scripts[1] );
```

As the second step in using the APIs, you would use methods in the `DKDocumentConverter` class to convert your XML documents. The methods perform the following tasks:

getSchemaMappingNames()

Retrieves the schema mapping names from the repository.

getXSLTQuery()

Retrieves the XSLT query script that can automatically convert XML documents from the user-defined schema to the storage schema.

transformXMLDocument()

Transforms an XML document using the XSLT query script that you retrieved.

deleteSchemaMapping()

Deletes a schema mapping from the repository.

Java

```
import com.ibm.mm.sdk.common.DKException;
import com.ibm.mm.sdk.cs.DKDatastoreICM;
import com.ibm.mm.sdk.xml.schema.DKDocumentConverter;
import com.ibm.mm.sdk.xml.schema.DKMapperException;
...
DKDatastoreICM cmDatastore = new DKDatastoreICM();
cmDatastore.connect(cmDatabase, cmUser, cmPassword, "");
System.out.println("MAPPING NAMES:");
Collection names=DKDocumentConverter.getSchemaMappingNames(cmDatastore);
System.out.println(names);
if (mapName == null)
    return;
String[] query=DKDocumentConverter.getXSLTQuery(cmDatastore, mapName);
System.out.println("XSLT Scripts for " + mapName);
if (query == null)
    System.out.println("NONE.");
else {
    for (int i = 0; i < query.length; i++) {
        if (i > 0)
            System.out.println("-----");
        System.out.println(query[i]);
    }
}

if (inputXMLDoc == null)
    return;
File inputFile = new File( inputXMLDoc );
File outputFile = new File( "APIoutput.xml");
DKDocumentConverter.transformXMLDocument( inputFile.toURL(),
query, outputFile );
System.out.println("Output in APIoutput.xml");
```

Programming runtime operations through the XML JavaBeans

The XML JavaBeans are Java classes that provide convenient interfaces to the DB2 Content Manager connector XML APIs and the DB2 Information Integrator for Content JavaBeans. They also serve as the communication layer between the Web services and the connector APIs.

The XML JavaBeans can perform runtime operations such as import, export, search, create, update, retrieve, delete, and document routing. They do not support system administration functions.

If you decide to program applications that communicate with DB2 Content Manager directly through the XML JavaBeans, then you can direct your XML requests straight to the CMBXMLMessage bean (similar to what the Web services would do). Your XML requests must follow the structure described in the cmbmessages.xsd schema.

The following JavaBean example sets up a CMBXMLMessage bean to send an XML search request directly to it:

Java

```
public class TXMLSearch2 {
    public static void main(String[] args) throws Exception {
        // Create beans
        CMBXMLServices xmlServices = new CMBXMLServices();
        // Create the search request message and get the reply message
        CMBXMLMessage reply = search(xmlServices, dstype, server,
            userid, password, entity, condition);
        System.out.println("Search reply: " + reply.getAsString());
    }
    static public CMBXMLMessage search(CMBXMLServices xmlServices,
        String dstype, String server, String userid, String password,
        String entity, String condition)
        throws CMBException, Exception
    {
        return search(xmlServices, dstype, server, userid, password,
            entity, condition, null);
    }
    static public CMBXMLMessage search(CMBXMLServices xmlServices,
        String dstype, String server, String userid, String password,
        String entity, String condition, String maxResults)
        throws CMBException, Exception
    {
        // Create the query string
        int queryType = CMBBaseConstant.CMB_QS_TYPE_XPATH;
        String queryString = "/" + entity;
        queryString += "[" + condition + "]";
        String connectString = "";
        // If the server name is followed by a parenthesized string,
        // use that string for the connect string.
        // e.g. ICMNLSDB(SCHEMA=ICMADMIN)
        if (server.indexOf("(") > 0) {
            connectString = server.substring(server.indexOf("(") + 1);
            server = server.substring(0, server.indexOf("("));
            if (connectString.endsWith("(")) {
                connectString = connectString.substring(0,
                    connectString.length() - 1);
            }
        }
    }
}
// continued...
```

Then to send an XML search request using the above example, you can pass in your server name (ICMNLSDDB in the example) and connectString as SCHEMA=ICMADMIN:

Java

```
StringBuffer XMLBuffer = new StringBuffer();
String maxResString = "";
if (maxResults != null) maxResString="maxResults=\\\\" +
maxResults + "\\\"";
XMLBuffer.append("<?xml version='1.0' encoding='UTF-8' ?>");
XMLBuffer.append("<RunQueryRequest " + maxResString +
" version='\" + CMBXMLConstant.CMB_LATEST_VERSION + \"'
" retrieveOption='\" + CMBXMLConstant.CMB_RETRIEVE_CONTENT +
"\" contentOption='\" + CMBXMLConstant.CMB_CONTENT_ATTACHMENTS +
"\" \" + TXMLTestcase.namespace + ">");
XMLBuffer.append("<AuthenticationData connectString='\" +
connectString + \"' configString='\">");
XMLBuffer.append("<ServerDef>");
XMLBuffer.append( "<ServerType>" + dstype + "</ServerType>");
XMLBuffer.append( "<ServerName>" + server + "</ServerName>");
XMLBuffer.append("</ServerDef>");
XMLBuffer.append("<LoginData>");
XMLBuffer.append( "<UserID>" + userid + "</UserID>");
XMLBuffer.append( "<Password>" + password + "</Password>");
XMLBuffer.append("</LoginData>");
XMLBuffer.append("</AuthenticationData>");
XMLBuffer.append("<QueryCriteria>");
XMLBuffer.append("<QueryString>" + queryString + "</QueryString>");
XMLBuffer.append("</QueryCriteria>");
XMLBuffer.append("</RunQueryRequest>");
System.out.println("XMLRequest: \n\n" + XMLBuffer.toString() + "\n\n");
CMBXMLMessage doc = new CMBXMLMessage(XMLBuffer.toString(), null);
// Search using the makeRequest method on the CMBXMLServices bean
System.out.println("Performing search");
System.out.println(XMLBuffer.toString());
CMBXMLMessage reply = xmlServices.makeRequest(doc);
System.out.println("Search reply");
System.out.println(reply.getAsString());
TXMLTestcase.printAttachments(reply.getAttachments());
return reply;
}
}
```

This section contains the following ways to create XML requests using the cmbmessages.xsd schema:

- “Listing DB2 Content Manager servers with ListServerRequest” on page 464
- “Authenticating Web service requests for security” on page 465 (always required)
- “Changing a password with XML requests” on page 466
- “Listing DB2 Content Manager entities with ListSchemaRequest” on page 466
- “Creating DB2 Content Manager items with CreateItemRequest” on page 468
- “Searching DB2 Content Manager items with RunQueryRequest” on page 469
- “Retrieving DB2 Content Manager items with RetrieveItemRequest” on page 472
- “Viewing your user privileges with XML requests” on page 476
- “Working with DB2 Content Manager folders through XML requests” on page 477
- “Updating DB2 Content Manager items with an XML UpdateItemRequest” on page 480
- “Deleting DB2 Content Manager items with DeleteItemRequest” on page 487

- “Checking DB2 Content Manager items out and in with CheckoutItemRequest and CheckinItemRequest” on page 488
- “Moving DB2 Content Manager items between entities with MoveItemRequest” on page 492
- “Linking DB2 Content Manager items with CreateLinkRequest or DeleteLinkRequest” on page 490
- “Accessing DB2 Content Manager document routing using XML-based requests” on page 493
- “Batching multiple requests in XML requests” on page 509

Listing DB2 Content Manager servers with ListServerRequest

To list all available DB2 Content Manager servers, create a ListServerRequest that identifies the following information (text in brackets is optional):

```
<ListServerRequest>
  [ <ServerType>ICM</ServerType> ]
</ListServerRequest>
```

<ListServerRequest> elements:

<ServerType> (optional)

Identifies the types of server definitions to list. If you do not specify a <ServerType>, then the request returns definitions of all DB2 Content Manager Version 8.3 servers known to the system.

The following example lists all servers of type ICM:

XML request

```
<ListServersRequest
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <ServerType>ICM</ServerType>
</ListServersRequest>
```

As a result, DB2 Content Manager returns an XML ListServerReply that returns <ServerDef> objects for all available servers.

XML reply

```
<ListServersReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
  <ServerDef>
    <ServerType>ICM</ServerType>
    <ServerName>icmn1sdb</ServerName>
  </ServerDef>
  <ServerDef>
    <ServerType>ICM</ServerType>
    <ServerName>CMA30</ServerName>
  </ServerDef>
  <ServerDef>
    <ServerType>ICM</ServerType>
    <ServerName>cma15</ServerName>
  </ServerDef>
</ListServersReply>
```

Authenticating Web service requests for security

Every time that you make a request to the Web services, you must pass in a DB2 Content Manager user ID and password, or a WebSphere credential token associated with a DB2 Content Manager user. If a user does not have the privilege to perform the specific request, then the request is not processed and an error is returned in the SOAP reply. For example, if a user wants to make change to an insurance policy, but only has view privileges, the user cannot make any changes to the policy.

Important: By default, the user ID and password passed in the Web services request are not encrypted. This is not a big issue if all of the Web service requests are being processed within the firewall. However, if the client is outside the firewall, you should use SSL to send your SOAP requests.

To authenticate your Web service requests, create an `AuthenticationData` object. You must then include this object in every request.

```
<AuthenticationData connectString="string"
  configString="string"
  [ connectToWorkflow="boolean" ] >
  <ServerDef>
    <ServerName>string</ServerName>
    [ <ServerType>ICM</ServerType> ]
  </ServerDef>
  <!-- You can specify either a user ID/password or a WebSphere SSO credential: -->
  <LoginData>
    <UserID>string</UserID>
    <Password>string</Password>
    <!-- or --> <Credential>base64Binary</Credential>
  </LoginData>
</AuthenticationData>
```

<AuthenticationData> elements:

<ServerDef> (required)

Identifies your content server's `<ServerName>` (for example, concord) and an optional `<ServerType>` (the default is ICM). To get this information, see "Listing DB2 Content Manager servers with ListServerRequest" on page 464.

<LoginData> (required)

Authenticates the user by either user ID/password or by a WebSphere SSO credential.

<AuthenticationData> attributes:

connectString (required)

Passes a server-specific property (beyond user ID and password) to establish a connection to the server. For the XML interface, always specify `SCHEMA=ICMADMIN`.

configString (required)

Passes a `CMBCConnection` property value to help construct a `dkDatastore` instance.

connectToWorkflow (optional)

Toggles an option to connect to the workflow server. The default is true.

Example:

```

<AuthenticationData connectionString="SCHEMA=ICMADMIN"
  configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icm1sdb</ServerName></ServerDef><LoginData>
  <UserID>testuser</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>

```

Changing a password with XML requests

To change your password in DB2 Content Manager, create a `ChangePasswordRequest` that identifies the following XML schema information:

```

<ChangePasswordRequest>
  <AuthenticationData> ... </AuthenticationData>
  <NewPassword>string</NewPassword>
</ChangePasswordRequest>

```

<ChangePasswordRequest> elements:

<AuthenticationData> (required)

Identifies the content server (`ServerDef`), a valid user ID (`UserID`), and password (`Password`)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<NewPassword> (required)

Specifies the new password to replace the old one in `<AuthenticationData>`.

The following example changes the password of user ID `testuser` to `passw0rd2`:

XML request

```

<ChangePasswordRequest
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <NewPassword>passw0rd2</NewPassword>
  <AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icm1sdb</ServerName></ServerDef><LoginData>
    <UserID>testuser</UserID><Password>passw0rd</Password></LoginData>
  </AuthenticationData>
</ChangePasswordRequest>

```

As a result, DB2 Content Manager returns an XML `ChangePasswordReply` that indicates success:

XML reply

```

<ChangePasswordReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
</ChangePasswordReply>

```

Listing DB2 Content Manager entities with ListSchemaRequest

To list all DB2 Content Manager entities where schemas are required, create a `ListSchemaRequest` that identifies the following information (attribute values in brackets are optional):

```

<ListSchemaRequest>
  <AuthenticationData> ... </AuthenticationData>
  <EntityList [ all="boolean" ]>
    [ <Entity [ name="string" /> ]
  </EntityList>
</ListSchemaRequest>

```

<ListSchemaRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<EntityList> (required)

Specifies which entities (for which schemas are required) to return. You can either set the all attribute to true to return all entities where schemas are required--or, you can specify the exact <Entity [name="string" /> objects to return.

The following example query lists all entities on a server that require a schema, and would additionally return an attachment of the full schema:

XML request

```

<ListSchemaRequest
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icm1sdb</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
  </AuthenticationData>
  <EntityList all="true"></EntityList>
</ListSchemaRequest>

```

As a result, DB2 Content Manager returns an XML ListSchemaReply that returns an EntityList of entities where schemas are required.

XML reply

```

<ListSchemaReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
  <EntityList>
    <Entity name="NOINDEX"/><Entity name="ICMBASE"/>
    <Entity name="ICMANNOTATION"/><Entity name="ICMNOTELOG"/>
    <Entity name="ICMSAVEDSEARCH"/><Entity name="ICMFORMS"/>
    <Entity name="ICMDRFOLDERS"/><Entity name="CLAIM_1047"/>
    <Entity name="ICMBASETEXT"/><Entity name="ICMBASESTREAM"/>
    <Entity name="INSURED_1047"/><Entity name="AGENT_1047"/>
    <Entity name="CLAIM2_1047"/><Entity name="DOC26"/>
    <Entity name="XYZ_ClaimFolder"/><Entity name="XYZ_AdjReport"/>
    <Entity name="XYZ_AutoPhoto"/><Entity name="XYZ_ClaimForm"/>
    <Entity name="XYZ_InsPolicy"/><Entity name="XYZ_PolReport"/>
  </EntityList>
</ListSchemaReply>

```


Creating DB2 Content Manager items with CreateItemRequest

To create items in DB2 Content Manager, create a CreateItemRequest that identifies the following XML schema information (text in brackets is optional):

```
<CreateItemRequest>
  <AuthenticationData> ... </AuthenticationData>
  <Item><ItemXML> ... </ItemXML></Item>
</CreateItemRequest>
```

<CreateItemRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item> (required)

Specifies an item to create on the DB2 Content Manager server. For details on how to convert your item to XML, see Chapter 11, “Working with XML services (Java only),” on page 429.

The following example request creates a policy with one TIFF image attached, and a policy number of 57904965371.

XML request

```
<CreateItemRequest
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icmn1sdb</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
  </AuthenticationData>
  <Item><ItemXML><XYZ_InsPolicy XYZ_Street="532 Camino Viejo"
    XYZ_City="Marina" XYZ_State="CA" XYZ_ZIPCode="90546" XYZ_PolicyNum=
    "57904965371" xmlns=""><XYZ_Insured XYZ_InsrdfName="Edward"
    XYZ_InsrdfLName="Smith" /><XYZ_Insured XYZ_InsrdfName="Jennifer"
    XYZ_InsrdfLName="Smith" /><XYZ_VIN XYZ_VIN="ICLA44P5KL9876543" />
    <ICMBASE><resourceObject MIMEType="image/tiff"
    xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"><label
    name="policyForm" /></resourceObject></ICMBASE></XYZ_InsPolicy>
  </ItemXML></Item>
</CreateItemRequest>
```

As a result, DB2 Content Manager returns an XML CreateItemReply that contains a URI for the new policy. You can enter this URI in a Web browser to view the item's XML structure (**Restriction:** This would not work directly with the XML beans because the URI is just the PID).

XML reply

```
<CreateItemReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
  <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
    3 ICM8 icmn1sdb13 XYZ_InsPolicy59 26 A1001001A04I10B33015B9925018
    A04I10B33015B992501 14 1029&server=icmn1sdb&dsType=ICM"/>
  </CreateItemReply>
```

The following example request creates a claim (claim number 8-123456) with one TIFF and one JPEG image attached.

XML request

```
<CreateItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item><ItemXML><XYZ_ClaimForm XYZ_ClaimFName="Joannifer"
XYZ_ClaimLName="Smith" XYZ_ClaimNumber="8-123456" XYZ_DriversLic=
"B12004960" XYZ_InsrdfName="Nicholas" XYZ_InsrdlName=
"Smith" XYZ_PolicyNum="57904965371" XYZ_IncDate="2001-01-20">
<ICMBASE><resourceObject MIMEType="image/tiff"
xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"><label
name="claimForm" /></resourceObject></ICMBASE>
<ICMBASE><resourceObject MIMEType="image/jpeg"
xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"><label
name="claimPhoto"/></resourceObject></ICMBASE>
</XYZ_ClaimForm>
</ItemXML></Item>
</CreateItemRequest>
```

As a result, DB2 Content Manager returns an XML CreateItemReply that contains a URI for the new claim. You can enter this URI in a Web browser to view the item's XML structure (**Restriction:** This would not work directly with the XML beans because the URI is just the PID).

XML reply

```
<CreateItemReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
3 ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B33016C1746518
A04I10B33016C174651 14 1027&amp;server=icmnlsdb&amp;dsType=ICM"/>
</CreateItemReply>
```

Searching DB2 Content Manager items with RunQueryRequest

To search for specific DB2 Content Manager items and retrieve Web links to them, create a RunQueryRequest that identifies the following XML schema information (text in brackets is optional):

```
<RunQueryRequest xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema"
[ retrieveOption="IDONLY" contentOption="URL" maxResults="integer"
version="LATEST_VERSION" ]>
  <AuthenticationData> ... </AuthenticationData>
  <QueryCriteria [ queryType="XPath" ]>
    <QueryString>string</QueryString>
  </QueryCriteria>
</RunQueryRequest>
```

<RunQueryRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and

password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<QueryCriteria> (required)

Specifies search parameters for the item or items that you want to retrieve in the XPATH (default) query syntax. For example, <QueryString>/XYZ_InsPolicy[@XYZ_PolicyNum="47809425673"]</QueryString>. For more information about query syntax, see “Example searches using the query language” on page 196.

<RunQueryRequest> attributes:

retrieveOption (optional)

Limits the information in your search results to improve the response time. Note that the more content that you request, the slower that your search will perform. Can have one of the following values:

IDONLY

Only returns the item IDs, and yields the fastest performance. This is the default. For example, an IDONLY QueryString of /XYZ_InsPolicy[@XYZ_PolicyNum="47809425673"] would search for all XYZ insurance policies with a policy number of 47809425673.

ATTRONLY

Returns the item IDs and all attribute values associated with them.

ITEMTREE

Returns the entire tree of information, including item IDs, attribute values, children, sub-children, and metadata. ITEMTREE does not retrieve links.

CONTENT

Returns all ITEMTREE information plus all of the content. The content returns as a URL attachment, depending on what you specify in the contentOption attribute. This yields the slowest performance.

contentOption (optional)

Specifies whether to return the item content as a Web address link or a binary attachment. Only applies when the retrieveOption is set to CONTENT. Can have one of the following values:

URL

Requests that a Web address to the item content (on the resource manager) be embedded in the XML description of the item. This is the default.

ATTACHMENTS

Requests that the item content be returned as a binary attachment in the reply (equivalent to the CMBXMLAttachment Java bean).

maxResults (optional)

Limits the number of search results to return. The default value is unlimited (0).

version (optional)

Specifies the version of the items to search for. For example, ALL_VERSIONS (all versions of the item), or specific version such as 1 or 2. The default value is LATEST_VERSION (the most recent version of the item).

The following example query requests a list of all policies:

XML request

```
<RunQueryRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema"
maxResults="0" version="latest-version(.)" contentOption="URL"
retrieveOption="ITEMTREE">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icmnlbdb</ServerName></ServerDef><LoginData>
      <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
    </AuthenticationData>
  <QueryCriteria>
    <QueryString>/XYZ_InsPolicy[ @XYZ_State LIKE "%"]</QueryString>
  </QueryCriteria>
</RunQueryRequest>
```

As a result, DB2 Content Manager returns an XML RunQueryReply that contains a list of Web addresses for all XYZ insurance policies. You can enter the URI in a Web browser to view an item's XML structure (**Restriction:** This would not work directly with the XML beans because the URI is just the PID).

XML reply

```
<RunQueryReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
  <ResultSet count="1">
    <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
      3 ICM8 icmnlbdb13 XYZ_InsPolicy59 26 A1001001A04I10B33609D5857118
      A04I10B33609D585711 14 1029&server=icmnlbdb&dsType=ICM">
      <ItemXML><XYZ_InsPolicy XYZ_City="Marina" XYZ_PolicyNum="57904965371"
        XYZ_State="CA" XYZ_Street="532 Camino Viejo" XYZ_ZIPCode="90546"
        cm:PID="93 3 ICM8 icmnlbdb13 XYZ_InsPolicy59 26
        A1001001A04I10B33609D5857118 A04I10B33609D585711 14 1029"
        xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema">
          <cm:properties type="document">
            <cm:lastChangeUserid value="ICMADMIN"/>
            <cm:lastChangeTime value="2004-09-10T20:36:09.462"/>
            <cm:createUserid value="ICMADMIN"/>
            <cm:createTime value="2004-09-10T20:36:09.462"/>
            <cm:semanticType value="1"/>
            <cm:ACL name="XYZInsurancePolicyACL"/>
            <cm:lastOperation name="RETRIEVE" value="SUCCESS"/>
            <cm:lastRetrieveOption value="32"/></cm:properties>
            <XYZ_Insured XYZ_InsrdFName="Edward" XYZ_InsrdLName="Smith"
              cm:PID="91 3 ICM8 icmnlbdb11 XYZ_Insured59 26
              A1001001A04I10B33609D5857118 A04I10B33609E627501 14 1030"/>
            <XYZ_Insured XYZ_InsrdFName="Jennifer" XYZ_InsrdLName="Smith"
              cm:PID="91 3 ICM8 icmnlbdb11 XYZ_Insured59 26
              A1001001A04I10B33609D5857118 A04I10B33609E637451 14 1030"/>
            <XYZ_VIN XYZ_VIN="ICLA44P5KL9876543" cm:PID="86 3 ICM8 icmnlbdb7
              XYZ_VIN59 26 A1001001A04I10B33609D5857118 A04I10B33609E646691 14
              1031"/></XYZ_InsPolicy>
          </ItemXML></Item></ResultSet>
    </Item>
  </ResultSet>
</RunQueryReply>
```

You can then program your custom application to display these Web links.

Retrieving DB2 Content Manager items with RetrieveItemRequest

To search for specific DB2 Content Manager items and retrieve them as either Web links or binary attachments, create a RetrieveItemRequest that identifies the following XML schema information (text in brackets is optional):

```
<RetrieveItemRequest retrieveOption="string"
[ contentOption="URL" version="LATEST_VERSION"
  checkout="false"> ]>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string" [ version="LATEST_VERSION" ] />
</RetrieveItemRequest>
```

<RetrieveItemRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item> (required)

Specifies the item to retrieve on the DB2 Content Manager server. Can contain the following attributes:

URI (required)

Specifies the identifier or the PID of the item to retrieve. You can obtain the URI through a RunQueryRequest. It also appears in the CreateItemReply.

version (optional)

Specifies the version of the items to retrieve. For example, ALL_VERSIONS (all versions of the item), or specific version such as 1 or 2. The default value is LATEST_VERSION (the most recent version of the item).

<RetrieveItemRequest> attributes:

retrieveOption (required)

Limits the information in your search results to improve the response time. Can have one of the following values:

ITEMTREE

Returns the entire tree of information, including item IDs, attribute values, children, sub-children, and metadata. ITEMTREE does not retrieve links.

CONTENT

Returns all ITEMTREE information plus all of the content. The content returns as a URL attachment, depending on what you specify in the contentOption attribute.

CONTENT_WITH_LINKS

Returns all ITEMTREE information, all of the content, and all of the item's inbound and outbound link information. The link information appears in the XML expression of the item. This yields the slowest performance.

contentOption (optional)

Specifies whether to return the item content as a Web address link or a

binary attachment. Only applies when the retrieveOption is set to CONTENT or CONTENT_WITH_LINKS. Can have one of the following values:

URL

Requests that a Web address to the item content (on the resource manager) be embedded in the XML description of the item. This is the default.

ATTACHMENTS

Requests that the item content be returned as a binary attachment in the reply (equivalent to the CMBXMLAttachment Java bean).

version (optional)

Specifies the version of the item to retrieve. For example, ALL_VERSIONS (all versions of the item), or specific version such as 1 or 2. The default value is LATEST_VERSION (the most recent version of the item).

checkout (optional)

Toggles whether to check out the item and lock it. The default value is false.

The following example query retrieves the 57904965371 policy and its content part URL location by specifying its item URI (as returned from the query search):

XML request

```
<RetrieveItemRequest contentOption="URL" retrieveOption="CONTENT"
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
      <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
    </AuthenticationData>
  <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
    3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B33015B9925018
    A04I10B33015B992501 14 1029&server=icmnlsdb&dsType=ICM"/>
</RetrieveItemRequest>
```

As a result, DB2 Content Manager returns an XML RetrieveItemReply that contains the 57904965371 policy and a URL for its content part:

XML reply

```
<RetrieveItemReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3
ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B33015B9925018
A04I10B33015B992501 14 1029&server=icmnlsdb&dsType=ICM">
<ItemXML><XYZ_InsPolicy XYZ_City="Marina" XYZ_PolicyNum="57904965371"
XYZ_State="CA" XYZ_Street="532 Camino Viejo XYZ_ZIPCode="90546"
cm:PID="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B33015B9925018 A04I10B33015B992501 14 1029"
xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<cm:properties type="document">
  <cm:lastChangeUserid value="ICMADMIN"/>
  <cm:lastChangeTime value="2004-09-10T20:30:15.363"/>
  <cm:createUserid value="ICMADMIN"/>
  <cm:createTime value="2004-09-10T20:30:15.363"/>
  <cm:semanticType value="1"/>
  <cm:ACL name="XYZInsurancePolicyACL"/>
  <cm:lastOperation name="RETRIEVE" value="SUCCESS"/>
  <cm:lastRetrieveOption value="288"/>
</cm:properties>
<XYZ_Insured XYZ_InsrdFName="Edward" XYZ_InsrdLName="Smith"
cm:PID="91 3 ICM8 icmnlsdb11 XYZ_Insured59 26
A1001001A04I10B33015B9925018 A04I10B33015D946781 14 1030"/>
<XYZ_Insured XYZ_InsrdFName="Jennifer" XYZ_InsrdLName="Smith"
cm:PID="91 3 ICM8 icmnlsdb11 XYZ_Insured59 26
A1001001A04I10B33015B9925018 A04I10B33015D955181 14 1030"/>
<XYZ_VIN XYZ_VIN="ICLA44P5KL9876543" cm:PID="86 3 ICM8 icmnlsdb7
XYZ_VIN59 26 A1001001A04I10B33015B9925018 A04I10B33015D968631
14 1031"/><ICMBASE cm:PID="85 3 ICM8 icmnlsdb7 ICMBASE58 26
A1001001A04I10B33015B9019018 A04I10B33015B901901 13 300"
cm:partNumber="1">
<cm:properties type="item" xsi:type="cm:partPropertyType">
  <cm:lastChangeUserid value="ICMADMIN"/>
  <cm:lastChangeTime value="2004-09-10T20:30:15.363"/>
  <cm:createUserid value="ICMADMIN"/>
  <cm:createTime value="2004-09-10T20:30:15.363"/>
  <cm:lastOperation name="RETRIEVE" value="SUCCESS"/>
  <cm:lastRetrieveOption value="1"/>
  <cm:ACL name="XYZInsurancePolicyACL"/>
  <cm:semanticType value="128"/>
</cm:properties>
<cm:resourceObject MIMEType="image/tiff" RMName="rmdb" SMSCollName=
"CBR.CLCT001" externalObjectName=" " resourceFlag="2" size="104137">
  <cm:URL value=
"http://hostname:9081/icrm/ICMResourceManager?order=
retrieve&item-id=A1001001A04I10B33015B90190&version=
1&collection=CBR.CLCT001&libname=icmnlsdb&update-date=
2004-09-10+20%3A30%3A15.486558&token=A4E6.EIGlq_6_csMMi058LhE;
&content-length=0"/>
</cm:resourceObject>
</ICMBASE></XYZ_InsPolicy>
</ItemXML></Item>
</RetrieveItemReply>
```

The following example query retrieves the 8-123456 claim with binary content part attached by specifying its item URI (as returned from the query search):

XML request

```
<RetrieveItemRequest contentOption="ATTACHMENTS"
retrieveOption="CONTENT"
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3
ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B33016C1746518
A04I10B33016C174651 14 1027&server=icmnlsdb&dsType=ICM"/>
</RetrieveItemRequest>
```

As a result, DB2 Content Manager returns an XML RetrieveItemReply that contains the 8-123456 claim and binary attachments for its content parts:

XML reply

```
<RetrieveItemReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3
ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B33016C1746518
A04I10B33016C174651 14 1027&server=icmnlsdb&dsType=ICM">
<ItemXML><XYZ_ClaimForm XYZ_ClaimFName="Joannifer"
XYZ_ClaimLName="Smith" XYZ_ClaimNumber="8-123456"
XYZ_DriversLic="B12004960" XYZ_IncDate="2001-01-20"
XYZ_InsrdfName="Nicholas" XYZ_InsrdfLName="Smith"
XYZ_PolicyNum="57904965371" cm:PID="93 3 ICM8 icmnlsdb13
XYZ_ClaimForm59 26 A1001001A04I10B33016C1746518
A04I10B33016C174651 14 1027" xmlns:cm=
"http://www.ibm.com/xmlns/db2/cm/api/1.0/schema" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance">
  <cm:properties type="document">
    ...
  </cm:properties>
  <ICMBASE cm:PID="85 3 ICM8 icmnlsdb7 ICMBASE58 26
A1001001A04I10B33016C1816218 A04I10B33016C181621 13 300"
cm:partNumber="1">
    <cm:properties type="item" xsi:type="cm:partPropertyType">
      ...
    </cm:properties>
    <cm:resourceObject MIMEType="image/tiff" RMName="rmdb"
      SMSCollName="CBR.CLLCT001" externalObjectName=" "
      resourceFlag="2" size="61578"> <cm:label name=
"A1001001A04I10B33016C18162A04I10B33016C181621"/>
    </cm:resourceObject>
  </ICMBASE>
```


XML reply (continued)

```
<ICMBASE cm:PID="85 3 ICM8 icmnlbdb7 ICMBASE58 26
A1001001A04I10B33016C1887218 A04I10B33016C188721 13 300"
cm:partNumber="2">
<cm:properties type="item" xsi:type="cm:partPropertyType">
...
</cm:properties>
<cm:resourceObject MIMEType="image/jpeg" RMName="rmdb"
SMSCollName="CBR.CLLCT001" externalObjectName=" "
resourceFlag="6" size="0"/>
</ICMBASE></XYZ_ClaimForm>
</ItemXML></Item>
</RetrieveItemReply>
```

Viewing your user privileges with XML requests

To view your privileges on a specific DB2 Content Manager item (including user privileges), create a `GetPrivilegesRequest` that identifies the following XML schema information:

```
<GetPrivilegesRequest>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string" />
</GetPrivilegesRequest>
```

<GetPrivilegesRequest> elements:

<AuthenticationData> (required)

Identifies the content server (`ServerDef`), a valid user ID (`UserID`), and password (`Password`)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item URI="string"/> (required)

Specifies the persistent identifier for the item that you want to list privileges for. You can obtain the URI through a `RunQueryRequest`. It also appears in the `CreateItemReply`.

The following example request lists all privileges for the 57904965371 policy by specifying its URI (as returned from the query search):

XML request

```
<GetPrivilegesRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema" >
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlbdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="90 3 ICM8 icmnlbdb10 CLAIM 104759 26
A1001001A04I13B04803F2085118 A04I13B04803F208511 14 1007"/>
</GetPrivilegesRequest>
```

As a result, DB2 Content Manager returns an XML `GetPrivilegesReply` that returns a `<Privileges>` list of all privileges associated with the item. Each `<Privilege>` object contains the name (for example, `VIEW_CONTENT` or `MODIFY_CONTENT`) and your authorized status, i.e., yes, no, or unknown.

XML reply

```
<GetPrivilegesReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Privileges>
  <Privilege authorized="YES" name="VIEW_CONTENT"/>
  <Privilege authorized="NO" name="MODIFY_CONTENT"/>
  <Privilege authorized="" name="EDIT_ATTRIBUTES"/>
  ...
</Privileges>
</GetPrivilegesReply>
```

Working with DB2 Content Manager folders through XML requests

A *folder* contains zero or more DB2 Content Manager items and zero or more folders. The object uses the following schema (text in brackets are optional):

```
<Folder>
[ <Item URI="string"><ItemXML> ... </ItemXML></Item> ]
[ <FolderItems>
  [ <Item URI="string"><ItemXML> ... </ItemXML></Item> ]
</FolderItems> ]
</Folder>
```

You can create a folder using `CreateItemRequest` and the `<cm:properties type="folder" xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"/>` setting (see the example near the end of this section).

You can request the following actions for folders in DB2 Content Manager.

Adding an item into a folder

```
<AddItemToFolderRequest [ newVersion="false"
checkin="true" checkout="true" ]>
  <AuthenticationData> ... </AuthenticationData>
  <Item>the item to add</Item>
  <Folder>the folder to add the item to</Folder>
</AddItemToFolderRequest>
```

The `AddItemToFolderRequest` can optionally use the `newVersion`, `checkout`, and `checkin` attributes (see “Updating DB2 Content Manager items with an XML `UpdateItemRequest`” on page 480 for details on these attributes).

Removing an item into a folder

```
<RemoveItemFromFolderRequest[ newVersion="false"
checkin="true" checkout="true" ]>
  <AuthenticationData> ... </AuthenticationData>
  <Item>the item to remove</Item>
  <Folder>the folder to remove the item from</Folder>
</RemoveItemFromFolderRequest>
```

The RemoveItemFromFolderRequest can optionally use the newVersion, checkout, and checkin attributes (see “Updating DB2 Content Manager items with an XML UpdateItemRequest” on page 480 for details on these attributes).

Retrieving all items inside of a folder

```
<RetrieveFolderItemsRequest [ retrieveOption="string"
contentOption="URL" version="LATEST_VERSION" ] >
  <AuthenticationData> ... </AuthenticationData>
  <Folder>the folder to retrieve the items from</Folder>
</RetrieveFolderItemsRequest>
```

The RetrieveFolderItemsRequest can optionally use the retrieveOption and contentOption attributes (see “Retrieving DB2 Content Manager items with RetrieveItemRequest” on page 472 for details on these attributes) along with maxItems to retrieve (the default is NO_MAX).

If successful, then DB2 Content Manager returns the following reply:

```
<RetrieveFolderItemsReply>
  <FolderItems>
    <Item> ... </Item>
    ...
  </FolderItems>
</RetrieveFolderItemsReply>
```

Retrieving all folders that an item is in

```
<RetrieveFoldersForItemRequest
[ retrieveOption="string" contentOption="URL" ] >
  <AuthenticationData> ... </AuthenticationData>
  <Item>the item to retrieve the folders for</Item>
</RetrieveFoldersForItemRequest>
```

The RetrieveFoldersForItemsRequest can optionally use the retrieveOption and contentOption attributes. See “Retrieving DB2 Content Manager items with RetrieveItemRequest” on page 472 for details on these attributes.

If successful, then DB2 Content Manager returns the following reply:

```
<RetrieveFoldersForItemReply>
  <Folder> ... </Folder>
  ...
</RetrieveFoldersForItemReply>
```

The following example creates a folder and puts the 57904965371 policy into it by specifying the item URI (as returned from the query search). **Important:** You must

specify `<cm:properties type="folder" xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"/>` to categorize the new item as a folder.

XML request

```
<CreateItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item><ItemXML><XYZ_InsPolicy XYZ_Street="532 Camino Viejo"
  XYZ_City="Marina" XYZ_State="CA" XYZ_ZIPCode="90546"
  XYZ_PolicyNum="57904965371" xmlns="">
  <cm:properties type="folder" xmlns:cm=
    "http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"/>
  </XYZ_InsPolicy>
</ItemXML></Item>
</CreateItemRequest>
```

As a result, DB2 Content Manager returns an XML `CreateItemReply` that contains the URI for the new folder:

XML reply

```
<CreateItemReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
  3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B45558I8302118
  A04I10B45558I830211 14 1029&server=icmnlsdb&dsType=ICM"/>
</CreateItemReply>
```

The following example adds the 8-123456 claim to the same folder by specifying their URIs:

XML request

```
<AddItemToFolderRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Folder URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
  3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B45558I8302118
  A04I10B45558I830211 14 1029&server=icmnlsdb&dsType=ICM"/>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
  3 ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B45555F4042518
  A04I10B45555F404251 14 1027&server=icmnlsdb&dsType=ICM"/>
</AddItemToFolderRequest>
```

As a result, DB2 Content Manager returns an XML `AddItemToFolderReply` that indicates success:

XML reply

```
<AddItemToFolderReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
</AddItemToFolderReply>
```

The following example deletes the folder by specifying its URI:

XML request

```
<DeleteItemRequest
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
      <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
  </AuthenticationData>
  <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDurl?pid=93
    3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B45558I8302118
    A04I10B45558I830211 14 1029&server=icmnlsdb&dsType=ICM" />
  </DeleteItemRequest>
```

As a result, DB2 Content Manager returns an XML DeleteItemReply that indicates success:

XML reply

```
<DeleteItemReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
</DeleteItemReply>
```

Updating DB2 Content Manager items with an XML UpdateItemRequest

To update items from DB2 Content Manager, create an UpdateItemRequest that identifies the following XML schema information:

```
<UpdateItemRequest [ newVersion="false"
  checkin="true" checkout="true" ]>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string">
    [ <XMLItem> ... </XMLItem> ]
  </Item>
</UpdateItemRequest>
```

<UpdateItemRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item URI="string"> (required)

Specifies the item to update in the DB2 Content Manager server.

| **<XMLItem> (optional)**

| Replaces the entire item's XML specification, including all its metadata,
| attributes, and child components. For details on how to convert your item
| to XML, see Chapter 11, "Working with XML services (Java only)," on page
| 429.

| The following example replaces the XYZ_InsPolicy item specified in the
| insPolicy and attachments variables.

| **<UpdateItemRequest> attributes:**

| **newVersion (optional)**

| Specifies whether to increment the version number on the item that you
| update. The default value is false.

| **checkout (optional)**

| Toggles whether to check the item out (thus locking it) before performing
| the update request on it. The default value is true.

| **checkin (optional)**

| Toggles whether to check an item in after performing the update request
| on it. The default value is true. If the item is not checked out then this
| setting fails.

| You can specify the following types of updates:

- | • "Adding objects inside DB2 Content Manager items with an XML
| UpdateItemRequest" on page 482
- | • "Replacing objects inside DB2 Content Manager items with an XML
| UpdateItemRequest" on page 484
- | • "Deleting objects inside DB2 Content Manager items with an XML
| UpdateItemRequest" on page 486

| The following example batches all three updates together to update the
| 57904965371 policy:

XML request

```
<UpdateItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029" />
<Add>
  <ChildItem parentURI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029">
    <ItemXML><XYZ_Insured XYZ_InsrdFName="Alice"
XYZ_InsrdLName="Smith" /></ItemXML></ChildItem>
</Add>
<Replace>
  <Attribute name="XYZ_Street"><Value>123 Cedar Rd</Value></Attribute>
  <Attribute name="XYZ_ZIPCode"><Value>90543</Value></Attribute>
  <ChildItem parentURI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029">
    <ItemXML><XYZ_VIN
xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
XYZ_VIN="1RI035P5RU5435209" cm:PID="86 3 ICM8 icmnlsdb7
XYZ_VIN59 26 A1001001A04I10B35710G9498818 A04I10B35710J581901
14 1031" /></ItemXML>
  </ChildItem>
  <Content PID="85 3 ICM8 icmnlsdb7 ICMBASE58 26
A1001001A04I10B35710G9582118 A04I10B35710G958211 13 300">
    <Attachment content-id="policyForm"/>
  </Content>
</Replace>
<Delete>
  <ChildItem URI="91 3 ICM8 icmnlsdb11 XYZ_Insured59 26
A1001001A04I10B35710G9498818 A04I10B35710J374681 14 1030" />
</Delete>
</UpdateItemRequest>
```

As a result, DB2 Content Manager returns an XML UpdateItemReply that contains the URI for the updated policy:

XML reply

```
<UpdateItemReply xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=84
3 ICM8 icmnlsdb5 DOC2659 26 A1001001A04I02B22524G4418418
A04I02B22524G441841 14 1019&server=icmnlsdb&dsType=ICM"/>
</UpdateItemReply>
```

Adding objects inside DB2 Content Manager items with an XML UpdateItemRequest

Using the Add element in an UpdateItemRequest, you can add the following types of XML schema values to existing DB2 Content Manager items (text in brackets is optional):

```
<Add>
[ <ChildItem parentURI="string"><ItemXML> ... </ItemXML></ChildItem> ]
[ <Content [ index="int" ] >
  <attachment content-id="string">...</attachment>
```

```

|         <!-- or --> <URL>...</URL>
|     </Content> ]
|     [ <Annotation [ index="string" ] >
|         <attachment content-id="string">...</attachment>
|         <!-- or --> <URL>...</URL>
|         </Annotation> ]
|     [ <Notelog [ index="int" ]>
|         <attachment content-id="string">...</attachment>
|         <!-- or --> <URL>...</URL>
|         </Notelog> ]
|     [ <Part attrName="string">
|         <attachment content-id="string">...</attachment>
|         <!-- or --> <URL>...</URL>
|         </Part> ]
| </Add>

```

<ChildItem> (optional)

Adds an existing child item to an existing child component collection with whatever you specify in ItemXML. For details on how to convert your item to XML, see Chapter 11, “Working with XML services (Java only),” on page 429. The attributes include:

parentURI (required)

The persistent identifier of the parent item to add the child item to. You can obtain the URI through a RunQueryRequest. It also appears in the CreateItemReply.

<Content> (optional)

Adds an attachment or URL as a content part to the item. This is the same as an ICM base part. The attributes include:

index (optional)

Ranks where to add the part. The default is to add the part at the end.

<Notelog> (optional)

Adds a note log to the item, and can include an attachment or URL. The attributes include:

index (optional)

Ranks where to add the annotation. The default is to add the notelog at the end.

<Annotation> (optional)

Adds an annotation to the item and can include an attachment or a URL. The attributes include:

index (optional)

Ranks where to add the annotation. The default is to add the annotation at the end.

<Part> (optional)

Adds your own user-defined part with the attribute name that you specify. Can include either an attachment or URL. The attributes include:

attrName (required)

Specifies the name of the user-defined attribute to add.

The following example adds a child item to the 57904965371 policy into it by specifying their item URIs (as returned from the query search):

XML request

```
<UpdateItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
  </AuthenticationData>
<Item URI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029" />
<Add>
  <ChildItem parentURI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029">
    <ItemXML><XYZ_Insured XYZ_InsrdFName="Alice"
      XYZ_InsrdLName="Smith" /></ItemXML></ChildItem>
  </Add>
</UpdateItemRequest>
```

As a result, DB2 Content Manager returns an XML UpdateItemReply that contains the URI for the updated policy:

XML reply

```
<UpdateItemReply xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=84
3 ICM8 icmnlsdb5 DOC2659 26 A1001001A04I02B22524G4418418
A04I02B22524G441841 14 1019&amp;server=icmnlsdb&amp;dsType=ICM"/>
</UpdateItemReply>
```

Replacing objects inside DB2 Content Manager items with an XML UpdateItemRequest

Using the Replace element in an UpdateItemRequest, you can overwrite the following types of XML schema values within existing DB2 Content Manager items:

```
<Replace>
[ <attribute> ... </attribute> ]
[ <ChildItem parentURI="string"><ItemXML> ... </ItemXML></ChildItem> ]
[ <Annotation PID="string">
  <attachment content-id="string">...</attachment>
  <!-- or --> <URL>...</URL>
</Annotation> ]
[ <Content PID="string">
  <attachment content-id="string">...</attachment>
  <!-- or --> <URL>...</URL>
</Content> ]
[ <Notelog PID="string">
  <attachment content-id="string">...</attachment>
  <!-- or --> <URL>...</URL>
</Notelog> ]
[ <Part PID="string" attrName="string">
  <attachment content-id="string">...</attachment>
  <!-- or --> <URL>...</URL>
</Part> ]
</Replace>
```

<Attribute> (optional)

Replaces attribute values.

| **<ChildItem> (optional)**

| Replaces a collection of child items. The attributes include:

| **parentURI (required)**

| The persistent identifier of the parent item to replace the child item
| in. You can obtain the URI through a RunQueryRequest. It also
| appears in the CreateItemReply.

| **<Annotation> (optional)**

| Replaces an annotation, and can include either an attachment or a URL.
| The attributes include:

| **PID (required)**

| Specifies the persistent identifier of the annotation.

| **<Content> (optional)**

| Replaces an item's content part with either the attachment or URL that you
| specify. This is the same as an ICM base part. The attributes include:

| **PID (required)**

| Specifies the persistent identifier of the content.

| **<Notelog> (optional)**

| Replaces the note log, and can include either an attachment or URL. The
| attributes include:

| **PID (required)**

| Specifies the persistent identifier of the note log.

| **<Part> (optional)**

| Replaces your user-defined part with the attribute name, attachment, or
| content URL that you specify. The attributes include:

| **PID (required)**

| Specifies the persistent identifier of the part.

| **attrName (required)**

| Specifies the name of the user-defined attribute to replace.

| The following example replaces two attributes, a child item, and a content part in
| the 57904965371 policy:

XML request

```
<UpdateItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029" />
<Replace>
  <Attribute name="XYZ_Street"><Value>123 Cedar Rd</Value></Attribute>
  <Attribute name="XYZ_ZIPCode"><Value>90543</Value></Attribute>
  <ChildItem parentURI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029">
    <ItemXML><XYZ_VIN
      xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
      XYZ_VIN="1RI035P5RU5435209" cm:PID="86 3 ICM8 icmnlsdb7
      XYZ_VIN59 26 A1001001A04I10B35710G9498818 A04I10B35710J581901
      14 1031" /></ItemXML>
    </ChildItem>
    <Content PID="85 3 ICM8 icmnlsdb7 ICMBASE58 26
      A1001001A04I10B35710G9582118 A04I10B35710G958211 13 300">
      <Attachment content-id="policyForm"/>
    </Content>
  </Replace>
</UpdateItemRequest>
```

As a result, DB2 Content Manager returns an XML UpdateItemReply that contains the URI for the updated policy:

XML reply

```
<UpdateItemReply xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=84
3 ICM8 icmnlsdb5 DOC2659 26 A1001001A04I02B22524G4418418
A04I02B22524G441841 14 1019&server=icmnlsdb&dsType=ICM"/>
</UpdateItemReply>
```

Deleting objects inside DB2 Content Manager items with an XML UpdateItemRequest

Using the Delete element in an UpdateItemRequest, you can delete the following types of XML schema values from existing DB2 Content Manager items (this only removes this item's references to the attribute values):

```
<Delete>
[ <ChildItem URI="string"> ... </ChildItem> ]
[ <Annotation PID="string"> ]
[ <Content PID="string"> ]
[ <Notelog PID="string"> ]
[ <Part PID="string" attrName="string"> ]
</Delete>
```

<ChildItem> (optional)

Removes but does not delete the child component from the collection.

<Annotation> (optional)

Deletes the annotation from the item PID attribute that you specify.

<Content> (optional)

Deletes the attachment from the item of the PID that you specify. This is the same as an ICM base part.

<Notelog> (optional)

Deletes the notelog from the item PID attribute that you specify.

<Part> (optional)

Deletes a user-defined part associated with the item PID and attribute name that you specify.

The following example deletes a child item from the 57904965371 policy into it by specifying their item URIs (as returned from the query search):

XML request

```
<UpdateItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
  </AuthenticationData>
  <Item URI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
    A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029" />
  <Delete>
    <ChildItem URI="91 3 ICM8 icmnlsdb11 XYZ_Insured59 26
      A1001001A04I10B35710G9498818 A04I10B35710J374681 14 1030" />
  </Delete>
</UpdateItemRequest>
```

As a result, DB2 Content Manager returns an XML UpdateItemReply that contains the URI for the updated policy:

XML reply

```
<UpdateItemReply xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDurl?pid=84
  3 ICM8 icmnlsdb5 DOC2659 26 A1001001A04I02B22524G4418418
  A04I02B22524G441841 14 1019&server=icmnlsdb&dsType=ICM"/>
</UpdateItemReply>
```

Deleting DB2 Content Manager items with DeleteItemRequest

To delete items from DB2 Content Manager, create a DeleteItemRequest that identifies the following information (attribute values in brackets are optional):

```
<DeleteItemRequest>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string" />
</DeleteItemRequest>
```

<DeleteItemRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item URI="string" /> (required)

Specifies the item to delete from the DB2 Content Manager server.

The following example deletes the 57904965371 policy by specifying its item URI (as returned from the query search):

XML request

```
<DeleteItemRequest xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
      <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
    </AuthenticationData>
    <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
      3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B44211H0825818
      A04I10B44211H082581 14 1029&server=icmnlsdb&dsType=ICM" />
  </DeleteItemRequest>
```

As a result, DB2 Content Manager returns an XML DeleteItemReply that indicates success:

XML reply

```
<DeleteItemReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
</DeleteItemReply>
```

Checking DB2 Content Manager items out and in with CheckoutItemRequest and CheckinItemRequest

When you check out items from DB2 Content Manager, the server locks the items with your user ID so that no other user can edit them. This prevents users from overriding each others' changes. It is especially useful in batch requests when you must edit certain items for a long period of time.

To check out items from DB2 Content Manager, create a CheckoutItemRequest that identifies the following information (attribute values in brackets are optional):

```
<CheckoutItemRequest>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string" />
</CheckoutItemRequest>
```

<CheckoutItemRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<Item URI="string" /> (required)

Specifies the item to check out from the DB2 Content Manager server.

The following example query checks out the 47809425673 policy by specifying its item URI:

XML request

```
<CheckoutRequest xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef>
      <ServerType>ICM</ServerType><ServerName>concord</ServerName>
    </ServerDef>
    <LoginData>
      <UserID>icmadmin</UserID><Password>ec1lent</Password>
    </LoginData>
  </AuthenticationData>
  <Item URI="http://icmserver/CMBGenericWebService/CMBGetPIDUrl?pid=92 3
  ICM7 concord13 XYZ_InsPolicy59 26 A1001001A04B25B14339H1785918
  A04B25B14339H178591 14 1048&server=concord&dsType=ICM" />
</CheckoutItemRequest>
```

As a result, DB2 Content Manager returns an XML CheckoutItemReply that indicates success.

XML reply

```
<CheckoutItemReply xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
</CheckoutItemReply>
```

To check items back into DB2 Content Manager, create a CheckinItemRequest that identifies the following information (attribute values in brackets are optional):

```
<CheckinItemRequest>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string" />
</CheckinItemRequest>
```

<CheckinItemRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item URI="*string*" />

Specifies the item to check into the DB2 Content Manager server.

The following example query checks in the 47809425673 policy by specifying its item URI:

XML request

```
<CheckinRequest xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>concord</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>ec1lent</Password></LoginData>
  </AuthenticationData>
  <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=92 3
    ICM7 concord13 XYZ_InsPolicy59 26 A1001001A04B25B14339H1785918
    A04B25B14339H178591 14 1048&server=concord&dsType=ICM" />
</CheckinItemRequest>
```

As a result, DB2 Content Manager returns an XML CheckinItemReply that indicates success.

XML reply

```
<CheckinItemReply xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
</CheckinItemReply>
```

Linking DB2 Content Manager items with CreateLinkRequest or DeleteLinkRequest

To create or delete links between items in DB2 Content Manager, create a CreateLinkRequest or a DeleteLinkRequest that identifies the following XML schema information (text in brackets is optional):

```
<CreateLinkRequest> <!-- or --> <DeleteLinkRequest>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string">
    <links>
      <inbound fromItem="URI" linkType="string"
        [ linkInfoItem="string" ] />
      <!-- or -->
      <outbound toItem="URI" linkType="string"
        [ linkInfoItem="string" ] />
    </links>
  </Item>
</CreateLinkRequest> <!-- or --> </DeleteLinkRequest>
```

<CreateLinkRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item> (required)

Specifies an item to create on the DB2 Content Manager server. For details on how to convert your item to XML, see Chapter 11, “Working with XML services (Java only),” on page 429.

<links> (required)

Specifies the <inbound> links (for linking from an item URI) or <outbound> links (for linking to an item URI) to create.

The following example creates an outbound link from the 8-123456 claim to the 57904965371 policy by specifying their item URIs (as returned from the query search):

XML request

```
<CreateLinkRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3
ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B44211H0825818
A04I10B44211H082581 14 1029&server=icmnlsdb&dsType=ICM">
<links xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema">
  <outbound toItem=
  "http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3 ICM8
icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B44213F1658218
A04I10B44213F165821 14 1027&server=icmnlsdb&dsType=ICM"
  linkType="Contains"/></links>
</Item>
</CreateLinkRequest>
```

As a result, DB2 Content Manager returns an XML CreateLinkReply that indicates success.

XML reply

```
<CreateLinkReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
</CreateLinkReply>
```

The following example deletes the outbound link from the 8-123456 claim to the 57904965371 policy by specifying their item URIs (as returned from the query search):

XML request

```
<DeleteLinkRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3
ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B44211H0825818
A04I10B44211H082581 14 1029&server=icmnlsdb&dsType=ICM">
<links xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema">
  <outbound toItem=
  "http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3 ICM8
icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B44213F1658218
A04I10B44213F165821 14 1027&server=icmnlsdb&dsType=ICM"
  linkType="Contains"/></links>
</Item>
</DeleteLinkRequest>
```


As a result, DB2 Content Manager returns an XML DeleteLinkReply that indicates success.

XML reply

```
<DeleteLinkReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
</DeleteLinkReply>
```

Moving DB2 Content Manager items between entities with MoveItemRequest

To move a DB2 Content Manager item from one entity to another (for modifying the itemtree of attribute and child values; not document content), create a MoveItemRequest that identifies the following information (attribute values in brackets are optional):

```
<MoveItemRequest [ checkout="true" checkin="true" ]>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI="string" />
  <NewValues newEntityName="string">
    <ItemXML><!-- Do not specify document content here --></ItemXML>
  </NewValues>
</MoveItemRequest>
```

<MoveItemRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Item URI="string"> (required)

Specifies the item to re-index.

<NewValues newEntityName="string"> (required)

Specifies the new value to set in the entity. The value must be an XML specification of the itemtree, conforming to the DB2 Content Manager data model.

<MoveItemRequest> attributes:

checkout (optional)

Toggles whether to check the item out (thus locking it) before performing the move request on it. The default value is true.

checkin (optional)

Toggles whether to check an item in after performing the move request on it. The default value is true. If the item is not checked out then this setting fails.

The following example re-indexes (moves) an item under CLAIM_1047 to CLAIM2_1047:

XML request

```
<MoveItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
  </AuthenticationData>
<Item URI="90 3 ICM8 icmnlsdb10 CLAIM_104759 26
A1001001A04I13B04803F2085118 A04I13B04803F208511 14 1007"/>
<NewValues newEntityName="CLAIM2_1047">
  <ItemXML>
    <CLAIM2_1047 xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
DESC_1047="This is a claim regarding the accident.">
      <cm:properties type="document"/>
      <VEHICLE2_1047 VIN_1047="38"></VEHICLE2_1047>
    </CLAIM2_1047>
  </ItemXML>
</NewValues>
</MoveItemRequest>
```

As a result, DB2 Content Manager returns an XML MoveItemReply that contains the brand-new URI (for CLAIM2_1047):

XML reply

```
<MoveItemReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="91 3 ICM8 icmnlsdb11 CLAIM2_104759 26
A1001001A04I13B04803F2085118 A04I16B41357H241001 14 1016"/>
</MoveItemReply>
```

Accessing DB2 Content Manager document routing using XML-based requests

DB2 Content Manager provides certain Web services that can help route your documents through a business process using the APIs.

Important: To define document routing processes in DB2 Content Manager Version 8.3, you must use the provided graphical process builder. You can run existing processes that were created using Version 8.2 APIs or Version 8.2 system administration client, but you cannot modify them or take advantage of Version 8.3 functionality (for example, split node, joint node, line-of-business node, decision point, and sub-process node) without using the graphical process builder. Furthermore, the processes that were created using the graphical process builder can run only on the Version 8.3 library server.

Caution: Creating document routing process with the Version 8.3 APIs risks unexpected behavior and damage to the system. The graphic builder averts this by validating a process before it is saved into the library server.

Listing work nodes with XML requests

A *work node* refers to a step at which work packages wait for actions to be taken by users or applications, or move ahead automatically. To list all of your DB2 Content Manager work nodes, create a `ListWorkNodesRequest` that contains the following information:

```
<ListWorkNodesRequest>
  <AuthenticationData> ... </AuthenticationData>
</ListWorkNodesRequest>
```

As a result, DB2 Content Manager returns an XML `ListWorkNodesReply` that contains all work nodes in the following format:

```
<WorkNode>
  <WorkNodeName>string</WorkNodeName>
  [ <Description>string</Description> ]
  <LongDescription>string</LongDescription>
  <ACLName>string</ACLName>
  [ <TimeLimit>minutes</TimeLimit> ]
  [ <OverLoadLimit>nonNegativeInteger</OverLoadLimit> ]
  [ <ActionListName>string</ActionListName> ]
  [ <ContainerDefinition> ... </ContainerDefinition> ]
  <WorkNodeExt><NameValuePair>
    <Name>attribute</Name><Value>string</Value>
  </NameValuePair>
  [ <NameValuePair> ... </NameValuePair> ]
</WorkNodeExt> ]
  <Type>0 | 1 | 6</Type> ]
  <!-- Type 0 (work basket) also requires: -->
  [ <EnterUserDLL>fileName</EnterUserDLL>
    <!-- or --> <EnterUserFunction>string</EnterUserFunction> ]
  [ <LeaveUserDLL>fileName</LeaveUserDLL>
    <!-- or --> <LeaveUserFunction>string</LeaveUserFunction> ]
  [ <OverloadUserDLL>fileName</OverloadUserDLL> ]
    <!-- or --> <OverloadUserFunction>string</OverloadUserFunction> ]
  <!-- Type 1 (collection pt), also requires: -->
  [ <CollectionResumeListEntry>
    <FolderItemTypeName>string</FolderItemTypeName>
    <RequiredItemTypeName>string</RequiredItemTypeName>
    <QuantityNeeded>int</QuantityNeeded>
  </CollectionResumeListEntry> ]
</WorkNode>
```

<WorkNodeName> (required)

Identifies what to call the work node.

<Description> (optional)

Summarizes the purpose of the work node, and displays in the system administration client. The maximum is 254 characters.

<LongDescription> (required)

Explains the work node in detail, and displays in the properties window. The maximum is 2048 characters.

<ACLName> (required)

Specifies the name of an access control list (ACL) to limit which users can access items in the work node.

<TimeLimit> (optional)

Specifies the minutes that can elapse before work packages expire in the work node (which then sets the notification flag to 1). The default value is 0 (no time limit).

<OverLoadLimit> (optional)

Specifies the maximum number of documents or folders that the work

node can contain before the system invokes the overload exit specified by the DLL or function. You can specify 0 if no limit is needed.

<ActionListName> (optional)

Specifies the list of actions that a user can perform on work items. You can assign an action list to each node in a process to specify the actions that the user can take at that step in the process.

<ContainerDefinition> (optional)

Defines a container that can carry variables from one work package to the next as the work package continues on the process. Can contain the following elements:

<ReadOnly>*false*</ReadOnly>

Indicates whether the work node can be updated. The default is false (can be updated). If set to true, then the work node becomes read-only.

<VariableName>*string*</VariableName>

Specifies the name for this work node variable. The limit is 32 characters.

<VariableType>0</VariableType>

Specifies the type of work node variable. Can be one of the following constants:

0 (CMB_ICM_TYPE_CHARACTER)

Character

1 (CMB_ICM_DR_WNV_TYPE_INTEGER)

Integer

3 (CMB_ICM_DR_WNV_TYPE_TIMESTAMP)

Timestamp

<VariableValue>*string*</VariableValue>

Specifies the value for this work node variable. The default limit is 254 characters.

<VariableLength>0</VariableLength>

Specifies the maximum length allowed for the <VariableValue> of this work node variable. This attribute is only valid when the "type" attribute is if the <VariableType> is set to 0. The default is 0 length.</xs:documentation>

<VariablePrompt>*string*</VariablePrompt>

Specifies the prompt for this work node variable. This is used to prompt the client user for updating the value attribute of this work node variable. The limit is 32 characters.

<Required>*false*</Required>

Indicates whether the <VariableValue> of this work node variable is required. The default is false. If set to true, then the client application should require the user to update the <VariableValue> of this work node variable.

<ShowToUser>*false*</ShowToUser>

Indicates whether this work node variable should be displayed to the client user. The default is false. If the value is set to true, then this work node variable should be displayed to the client user for updating.

<WorkNodeExt> (optional)

Extends a work node with user-defined attributes. This element can contain one or more <NameValuePair> elements for user-defined attributes.

<Type> (optional)

The work node type can be 0 (work basket), 1 (collection point), or 6 (business application node):

0 (work basket)

A step where the system keeps work packages that are in process or waiting to be processed. A work basket does not perform any actions on the content. When a user or application completes the required task on the work package, it is routed to the next work node. This is the default type of work node.

1 (collection point)

A special work node that does not correspond to a business task, and to which users do not have access. It represents an area where a folder waits for specified items (either other folders or documents) to be collected before continuing. The collection point routes the work package to another work node under two conditions: when the folder is complete, or when the time limit expires.

6 (business application node)

A process step where the library server invokes a user exit DLL to run lines of business applications. For details about user exits, see "Programming document routing user exits" on page 248.

All work baskets require a user exit to determine the tasks that a work package must complete when entering a work basket, leaving a work basket, or when the workbasket becomes full. The user exit can be either a dynamic link library (DLL) or function (depending on the programming language used). The selections are:

<EnterUserDLL> or <EnterUserFunction> (optional)

Specifies the DLL or function to call when a work package enters the work node.

<LeaveUserDLL> or <LeaveUserFunction> (optional)

Specifies the DLL or function to call when a work package leaves the work node.

<OverloadUserDLL> or <OverloadUserFunction> (optional)

Specifies the DLL or function to call when a work node reaches its overload limit.

All collection points require a <CollectionResumeListEntry> element to assign a list of item types that the folder item type must wait for before continuing on the process. The <CollectionResumeListEntry> requires the following elements within it:

<FolderItemTypeName> (required)

Identifies the name of the folder to gather items in. The work package must have this folder type associated with it, or the work package will just pass through this collection point.

<RequiredItemTypeName> (required)

Specifies the types of items to gather in the folder.

<QuantityNeeded> (required)

Specifies the amount of items need to complete the item type requirement.

Listing document processes with XML requests

A *process* is a series of steps that contains at least one start node, one end node, and one selection or action. To list all of your DB2 Content Manager processes, create a `ListProcessRequest` that contains the following information:

```
<ListProcessRequest>
  <AuthenticationData> ... </AuthenticationData>
</ListProcessRequest>
```

Tip: Alternatively, you can replace the `ListProcessRequest` tag with `ListProcessNamesRequest` if you only want a list of the process names.

As a result, DB2 Content Manager returns an XML `ListProcessReply` that contains all processes in the following format:

```
<Process>
  <ProcessName>string</ProcessName>
  <ACLName>string</ACLName>
  <Route>
    <RouteDefinition>
      <From>string</From><To>string</To>
      <RouteSelected>string</RouteSelected>
      [ <Extension><NameValuePair>
        <Name>attribute</Name><Value>string</Value>
      </NameValuePair> ... </Extension> ]
    </RouteDefinition>
    [ <RouteDefinition> ... </RouteDefinition> ]
  </Route>
  [ <Description>string</Description> ]
  [ <LongDescription>string</LongDescription> ]
  [ <TimeLimit>minutes</TimeLimit> ]
  [ <Extension><NameValuePair>
    <Name>attribute</Name><Value>string</Value>
  </NameValuePair>
  <NameValuePair> ... </NameValuePair> ]
  </Extension> ]
</Process>
```

<Process> elements:

<ProcessName> (required)

Identifies what to call the process.

<ACLName> (required)

Specifies the name of an access control list (ACL) to limit which users can access the process.

<Route> (required)

Defines the FROM node, the TO node, and the selection (which is a string). If selection is chosen, then the work packages moves to the work node defined in the TO.

<Description> (optional)

Summarizes the purpose of the process, and displays in the system administration client. The maximum is 254 characters.

<LongDescription> (optional)

Explains the process in detail, and displays in the properties window. The maximum is 2048 characters.

<TimeLimit> (optional)

Specifies the minutes that can elapse for the process to complete. When this expires, the notification flag in the work package is set to 1. The default value is 0 (no time limit).

<Extension> (optional)

Extends a process with user-defined attributes (appends more columns to the library server table). This element can contain one or more <NameValuePair> elements for user-defined attributes. Differs from container variables in that it is not carried with the work package.

Listing worklists with XML requests

A *worklist* contains an ordered list of work packages (documents or folders) that a user must complete. Worklists contain characteristics such as ordering (by priority or date), filtering (in suspend state or notify state), and quantity to return. These characteristics control how users see their work.

To list all of your DB2 Content Manager worklists, create a `ListWorkListsRequest` that contains the following information:

```
<ListWorkListsRequest>
  <AuthenticationData> ... </AuthenticationData>
</ListWorkListsRequest>
```

Tip: Alternatively, you can replace the `ListWorkListRequest` tag with `ListWorkListNamesRequest` if you only want a list of the worklist names.

As a result, DB2 Content Manager returns an XML `ListWorkListReply` that contains all processes in the following format:

```
<WorkList>
  <WorkListName>string</WorkListName>
  <ACLName>string</ACLName>
  <WorkNodeNames>string</WorkNodeNames>
  [ <Description>string</Description> ]
  [ <SelectionOrder>0 | 1</SelectionOrder> ]
  [ <SelectionFilterOnNotify>0 | 1 | 2</SelectionFilterOnNotify> ]
  [ <SelectionFilterOnSuspend>0 | 1 | 2</SelectionFilterOnSuspend> ]
  [ <SelectionFilterOnOwner>0 | 1</SelectionFilterOnOwner> ]
  [ <WorkPackagesToReturn>nonNegativeInteger</WorkPackagesToReturn> ]
</WorkList>
```

<WorkList> elements:

<WorkListName> (required)

Identifies what to call the work list.

<ACLName> (required)

Specifies the name of an access control list to limit which users can access work nodes and contained work items in the worklist.

<WorkNodeNames> (required)

Specifies which work nodes to include in the worklist by order of priority.

<Description> (optional)

Explains the work list.

<SelectionOrder> (optional)

Determines the order that work packages appear in the worklist. The choices are:

- 0** Sorts and returns the work packages by a user-defined priority.
- 1** Sorts and returns the work packages by the date that they were created.

<SelectionFilterOnNotify> (optional)

Filters the work packages by notify state. The choices are:

- 0 Only returns work packages that do not have the notify flag turned on.
- 1 Only returns work packages that have the notify flag turned on.
- 2 Returns work packages regardless of how the notify flag is set.

<SelectionFilterOnSuspend> (optional)

Filters the work packages by suspend state. The choices are:

- 0 Only returns work packages that do not have the suspend flag turned on.
- 1 Only returns work packages that have the suspend flag turned on.
- 2 Returns work packages regardless of how the suspend flag is set.

<SelectionFilterOnOwner> (optional)

Filters the work packages by what the user owns. The choices are:

- 0 Returns work packages regardless of who owns them.
- 1 Only returns work packages that the user owns.

<WorkPackagesToReturn> (optional)

Specifies the maximum number of work packages to return. The default value is 0 (all work packages returned).

Listing work packages with XML requests

A *work package* is a container for the items inside a worklist or process. To list all of your DB2 Content Manager work packages, create a `ListWorkPackagesRequest` that contains the following information (elements in brackets are optional):

```
<ListWorkPackagesRequest>
  <AuthenticationData> ... </AuthenticationData>
  <WorkListName>string</WorkListName>
  [ <WorkPackageOwner>string</WorkPackageOwner> ]
</ListWorkPackagesRequest>
```

<ListWorkPackagesRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<WorkListName> (required)

Specifies a worklist to list all of the work packages for.

<WorkPackageOwner> (optional)

Specifies an owner to list all of the work package for.

As a second option, you can replace the `ListWorkPackagesRequest` tag with the `ListNextWorkPackagesRequest` tag to return the next work package in the `<WorkListName>` (required) and by `<WorkPackageOwner>` (optional), and to check out the referenced item. If the item is hidden by a filter or already checked out by another user, then the request skips the item and moves to the next. If this request is called again and the list of work packages have remained the same, then the same work package is returned.

As a third option, you can replace the `ListWorkPackagesRequest` tag with the `ListWorkPackageCheckedOutOptionRequest` tag to return a work package by its URI (and to optionally check the item out):


```

<ListWorkPackageCheckedOutOptionRequest>
  <AuthenticationData> ... </AuthenticationData>
  <WorkPackageURI>string</WorkPackageURI>
  <CheckedOutItem>boolean</CheckedOutItem>
</ListWorkPackageCheckedOutOptionRequest>

```

<ListWorkPackageCheckedOutOptionRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<WorkPackageURI> (required)

Specifies the persistent identifier of the work package to list.

<CheckedOutItem> (required)

If set to true, then the item associated with the work package is checked out.

As a result of one of the three types of “list work package” requests, DB2 Content Manager returns the work package in an XML ListWorkPackagesReply, ListNextWorkPackagesReply, or ListWorkPackageCheckedOutOptionRequest using the following format:

```

<WorkPackage>
  <WorkPackageURI>aniURI</WorkPackageURI>
  <ItemIDURI>string</ItemIDURI>
  <ProcessName>string</ProcessName>
  <WorkNodeName>string</WorkNodeName>
  [ <WorkPackageOwner>string</WorkPackageOwner> ]
  [ <Priority>nonNegativeInteger</Priority> ]
  [ <SuspendState>false</SuspendState> ]
  [ <NotifyState>false</NotifyState> ]
  [ <NotifyTime>yyyy-mo-dd-hh.mi.ss</NotifyTime> ]
  [ <ResumeList><ResumeListDefinition>
    <RequiredItemTypeName>string</RequiredItemTypeName>
    <QuantityNeeded>0</QuantityNeeded>
  </ResumeListDefinition> ]
  </ResumeList> ]
  [ <ResumeTime>yyyy-mo-dd-hh.mi.ss</ResumeTime> ]
  [ <TimeLastMoved>yyyy-mo-dd-hh.mi.ss</TimeLastMoved> ]
  [ <UserLastMoved>string</UserLastMoved> ]
  [ <ProcessCompletionTime>yyyy-mo-dd-hh.mi.ss</ProcessCompletionTime> ]
  [ <ContainerData><NameValuePair>
    <Name>attribute</Name><Value>string</Value>
  </NameValuePair>
  [ <NameValuePair> ... </NameValuePair> ]
  </ContainerData> ]
</WorkPackage>

```

<WorkPackage> elements:

<WorkPackageURI> (required)

Indicates the persistent identifier of the work package.

<ItemIDURI> (required)

Indicates the persistent identifier of the item that the work package is associated with.

<ProcessName> (required)

Indicates the name of the process that the work package is in.

<WorkNodeName> (required)
Indicates the name of the work node that the work package is currently located at.

<WorkPackageOwner> (optional)
Indicates the owner of the work package.

<Priority> (optional)
Indicates a user-defined, arbitrary priority for the work package. For example, 1, 2, 3...

<SuspendState> (optional)
Indicates whether the work package has been suspended in a process.

<NotifyState> (optional)
Indicates whether the work package notifies users when it moves to another work node.

<NotifyTime> (optional)
Indicates the time (in military format: *yyyy-mo-dd-hh.mi.ss*) when the work package was moved into notify state.

<ResumeList> (optional)
Only applies to a work package folder, and requires a **<ResumeListDefinition>** tag. Keeps the process suspended until a certain item type (specified in **<RequiredItemTypeName>**) of a specific quantity (specified in **<QuantityNeeded>**) appears in the folder. The default **QuantityNeeded** is 0.

<ResumeTime> (optional)
Indicates the time (in military format: *yyyy-mo-dd-hh.mi.ss*) when the work package was moved into resume state.

<TimeLastMoved> (optional)
Indicates the time (in military format: *yyyy-mo-dd-hh.mi.ss*) when the work package was last moved.

<UserLastMoved> (optional)
Indicates the name of the user that last moved the work package.

<ProcessCompletionTime> (optional)
Totals the time (in military format: *yyyy-mo-dd-hh.mi.ss*) that the work package spent in the process.

<ContainerData> (optional)
Extends a work package with user-defined attributes. This element can contain one or more **<NameValuePair>** elements for user-defined attributes.

Listing actions in DB2 Content Manager with XML requests

An *action* specifies how a user can manipulate the work packages at a work node. To list the values of a particular DB2 Content Manager action, you can perform the following steps:

1. Create a **ListActionNamesRequest** that contains the following information (elements in brackets are optional) to return a list of all **ActionNames** inside of a particular **ActionList**:

```
<ListActionNamesRequest>
  <AuthenticationData> ... </AuthenticationData>
  <ActionListName>string</ActionListName>
</ListActionNamesRequest>
```

<ListActionRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<ActionListName> (required)

Specifies the name of the action list to list the action names for.

2. Create a ListActionRequest which contains the following information (elements in brackets are optional) to list the values of an ActionName:

```
<ListActionRequest>
  <AuthenticationData> ... </AuthenticationData>
  <ActionName>string</ActionName>
</ListActionRequest>
```

<ListActionRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<ActionName> (required)

Specifies the name of the action to list.

As a result, DB2 Content Manager returns an XML ListActionReply that contains all processes in the following format:

```
<ListActionReply>
  <ActionName>string</ActionName>
  <Predefined>boolean</Predefined>
  [ <AuditComment>string</AuditComment> ]
  [ <Description>string</Description> ]
  [ <Icon>string</Icon> ]
  [ <DisplayName>string</DisplayName> ]
  [ <Shortcut>string</Shortcut> ]
  [ <FunctionName>string</FunctionName> ]
  [ <ApplicationName>string</ApplicationName> ]
  [ <DLLName>string</DLLName> ]
</ListActionReply>
```

<Action> elements:

<ActionName> (required)

Specifies the name of the action.

<Predefined> (required)

If set to true, then this action is a DB2 Content Manager-defined action. If set to false, then this action is user-defined.

<AuditComment> (optional)

Specifies an audit trail comment.

<Description> (optional)

Explains the content in detail.

<Icon> (optional)

Specifies the location of the icon for the client application to display for this action. For example, C:\icon.gif.

<DisplayName> (optional)

Specifies the name for the client application to display as a menu choice to the user. For example, Insurance calculator.

| **<Shortcut> (optional)**

| Specifies keyboard keys for quick access to the action. For example,
| Ctrl+Shift+F4.

| **<FunctionName> (optional)**

| Specifies the function name in the DLL to call when the user selects this
| action. For example, WXV2UserExitSample. The function is run on the client.

| **<ApplicationName> (optional)**

| Specifies the user exit to call when the user selects this action. For example,
| C:\myapp.jsp. The exit is run on the client application.

| **<DLLName> (optional)**

| Specifies which link library DLL to call when the user selects this action.
| For example, C:\WXV2UserExitSample.dll. The DLL is run on the client
| application.

| **Updating a work package with UpdateWorkPackageRequest**

| To update certain objects inside a DB2 Content Manager work package, create an
| XML <UpdateWorkPackageRequest> that identifies the following information (text in
| brackets is optional):

| <UpdateWorkPackageRequest>
| <WorkPackageURI>string</WorkPackageURI>
| [<WorkPackageOwner>string</WorkPackageOwner>]
| [<Priority>nonNegativeInteger</Priority>]
| [<ContainerData><NameValuePair>
| <Name>attribute</Name><Value>string</Value>
| </NameValuePair>
| [<NameValuePair> ... </NameValuePair>]
| </ContainerData>]
| </UpdateWorkPackageRequest>

| **<UpdateWorkPackageRequest> elements:**

| **<WorkPackageURI> (required)**

| Specifies the persistent identifier of the work package to update.

| **<WorkPackageOwner> (optional)**

| Specifies the owner of the work package.

| **<Priority> (optional)**

| Specifies a user-defined, arbitrary priority for the work package. For
| example, 1, 2, 3...

| **<ContainerData> (optional)**

| Extends a work package with user-defined attributes. This element can
| contain one or more <NameValuePair> elements for user-defined attributes.

| The following example query updates user-defined ContainerData for the specified
| work package.

XML request

```
<UpdateWorkPackageRequest xmlns=
"http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>cmdb21s</ServerName></ServerDef>
    <LoginData><UserID>icmadmin</UserID><Password>o8rmond</Password>
    </LoginData>
  </AuthenticationData>
  <WorkPackageURI>89 3 ICM7 cmdb21s11 WORKPACKAGE58 26
  A1001001A04H17B32803I2340818 A04H17B33216H440631 03
  204</WorkPackageURI>
  <WorkPackageOwner>icmadmin</WorkPackageOwner>
  <Priority>2004</Priority>
  <ContainerData>
    <NameValuePair><NVPName>Loan amount</NVPName>
    <NVPValue>1000</NVPValue></NameValuePair>
    <NameValuePair><NVPName>last name </NVPName>
    <NVPValue>Latariot</NVPValue></NameValuePair>
  </ContainerData>
</UpdateWorkPackageRequest>
```

As a result, DB2 Content Manager returns an XML UpdateWorkPackageReply that indicates success.

Starting a document routing process with StartProcessRequest

To start a DB2 Content Manager through a document routing process, create a StartProcessRequest that identifies the following information (elements in brackets are optional):

```
<StartProcessRequest>
  <AuthenticationData> ... </AuthenticationData>
  <ProcessName>string</ProcessName>
  [ <WorkPackageOwner>string<WorkPackageOwner> ]
  [ <Priority>nonNegativeInteger</Priority> ]
  <ItemIDURI> ... </ItemIDURI>
  [ <ContainerData> ... </ContainerData> ]
</StartProcessRequest>
```

<StartProcessRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<ProcessName> (required)

Identifies what to call the process.

<WorkPackageOwner> (optional)

Specifies the owner of the work package.

<Priority> (optional)

Specifies a user-defined, arbitrary priority for the work package. For example, 1, 2, 3...

<ItemIDURI> (required)

Specifies the persistent identifier of the DB2 Content Manager item to start through the process.

<ContainerData> (optional)

Extends a work package with user-defined attributes. This element can contain one or more <NameValuePair> elements for user-defined attributes.

The following example query starts the process ClaimsProcess and assigns user-defined ContainerData to its work package.

XML request

```
<StartProcessRequest xmlns=
"http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN"
    configString=""><ServerDef>
    <ServerType>ICM</ServerType><ServerName>cmdb21s</ServerName>
    </ServerDef><LoginData><UserID>icmadmin</UserID>
    <Password>o8rmond</Password></LoginData>
  </AuthenticationData>
  <ProcessName>ClaimsProcess</ProcessName>
  <Priority>10</Priority>
  <ContainerData><NameValuePair><NVPName>Loan amount</NVPName>
    <NVPValue>1000</NVPValue></NameValuePair>
    <NameValuePair><NVPName>First name</NVPName>
    <NVPValue>Carly </NVPValue></NameValuePair>
    <NameValuePair><NVPName>Last name</NVPName>
    <NVPValue>Morreale</NVPValue></NameValuePair>
  </ContainerData>
  <ItemIDURI>86 3 ICM7 cmdb21s8 Claims2859 26
    A1001001A04H17B32800G0799718 A04H17B32800G079971
    14 1007</ItemIDURI>
</StartProcessRequest>
```

As a result, DB2 Content Manager returns an XML StartProcessReply that indicates success.

Ending a process with TerminateProcessRequest

You can explicitly terminate a process before it reaches the end node. This removes the work package (being routed) from the system. The item referenced in the work package is checked in if it is checked out. For parallel routes with many work packages, any one of these work packages can be used to terminate a process. To terminate a process, create a TerminateProcessRequest that identifies the following information:

```
<TerminateProcessRequest>
  <AuthenticationData> ... </AuthenticationData>
  <WorkPackageURI> ... </WorkPackageURI>
</TerminateProcessRequest>
```

<TerminateProcessRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<WorkPackageURI> (required)

Specifies the persistent identifier of the work package being routed by the process instance.

The following example query terminates the process for the specified work package.

XML request

```
<TerminateProcessRequest
xmlns="http://www.ibm.com/software/data/cmb/schemas/">
<AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlbdb</ServerName></ServerDef>
  <LoginData><UserID>icmadmin</UserID>
  <Password>aPassword</Password></LoginData>
</AuthenticationData>
<WorkPackageURI>89 3 ICM7 cmdb21s11 WORKPACKAGE58 26
A1001001A04H17B32803I2340818 A04H17B33216H440631 03
204</WorkPackageURI>
</TerminateProcessRequest>
```

As a result, DB2 Content Manager returns an XML `TerminateProcessReply` that indicates success. If the process has ended then there are no work package URIs returned.

Continuing a process with `ContinueProcessRequest`

Continues the work package (identified by the URI) to the next work node. This removes the specified work package from the system, and creates a new work package or many work packages (in the case of a parallel route). The item referenced by the item URI is checked in if it has been checked out. If the process has ended, then there are no work package URIs returned. To start a DB2 Content Manager item through the document routing process, create a `ContinueProcessRequest` that identifies the following information (elements in brackets are optional):

```
<ContinueProcessRequest>
  <AuthenticationData> ... </AuthenticationData>
  [ <WorkPackageURI>string</WorkPackageURI> ]
  <WorkPackageOwner>string</WorkPackageOwner>
  <RouteSelected> ... </RouteSelected>
  [ <ContainerData> ... </ContainerData> ]
</ContinueProcessRequest>
```

<ContinueProcessRequest> elements:

<AuthenticationData> (required)

Identifies the content server (`ServerDef`), a valid user ID (`UserID`), and password (`Password`)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<WorkPackageURI> (optional)

Identifies the persistent identifier of the work package to move to the next work node.

<WorkPackageOwner> (required)

Specifies the owner of the work package.

<RouteSelected> (required)

Specifies the name of the route that the process should take.

<ContainerData> (optional)

Extends a work package with user-defined attributes. This element can contain one or more `<NameValuePair>` elements for user-defined attributes.

The following example query assigns user-defined ContainerData to the specified work package, and continues it along the process.

XML request

```
<ContinueProcessRequest xmlns=
"http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>cmdb21s</ServerName></ServerDef>
    <LoginData><UserID>icmadmin</UserID>
    <Password>o8rmond</Password></LoginData>
  </AuthenticationData>
  <WorkPackageURI>89 3 ICM7 cmdb21s11 WORKPACKAGE58 26
    A1001001A04H17B32803I2340818 A04H17B33130C375901
    03 204</WorkPackageURI>
  <WorkPackageOwner>icmadmin</WorkPackageOwner>
  <RouteSelected>Accept</RouteSelected>
  <ContainerData><NameValuePair>
    <NVPName>Loan amount</NVPName>
    <NVPValue>1000</NVPValue></NameValuePair>
    <NameValuePair><NVPName>Last name</NVPName>
    <NVPValue>McKenna</NVPValue></NameValuePair>
  </ContainerData>
</ContinueProcessRequest>
```

As a result, DB2 Content Manager returns an XML ContinueProcessReply that indicates success.

Suspending a process with SuspendProcessRequest

You can suspend the process associated with a specific work package by sending a SuspendProcessRequest. This checks out the item in the work package, and keeps the work package's SuspendState attribute at true for a specific duration. For work package folders, you can define multiple ResumeListDefinitions that suspend the process until certain item types and quantities arrive.

To suspend a DB2 Content Manager document routing process, create a SuspendProcessRequest that identifies the following information (elements in brackets are optional):

```
<SuspendProcessRequest>
  <AuthenticationData> ... </AuthenticationData>
  <WorkPackageURI> ... <WorkPackageURI>
  [ <Duration>minutes</Duration> ]
  [ <ResumeListDefinition>
    <RequiredItemTypeName>string</RequiredItemTypeName>
    <QuantityNeeded>0</QuantityNeeded>
  </ResumeListDefinition> ]
</SuspendProcessRequest>
```

<SuspendProcessRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<WorkPackageURI> (required)

Specifies the persistent identifier of the work package in the process that you want to suspend.

<Duration> (optional)

Specifies how many minutes to suspend the process for. The default is 0.

<ResumeListDefinition> (optional)

Only applies to a work package folder. Keeps the process suspended until a certain item type (specified in <RequiredItemTypeName>) of a specific quantity (specified in <QuantityNeeded>) appears in the folder. The default QuantityNeeded is 0.

The following example query keeps the specified work package suspended in a process based on these rules:

- If the process is started with the Policy28 folder, then the work package is suspended for 30 minutes or whenever 5 items show up in the Policy28 folder (whichever occurs first).
- If the process is started with InsDoc28 folder, then the work package is suspended for 30 minutes or whenever 25 items show up in the InsDoc28 folder (whichever occurs first).
- If the process is started with a document or folder other than Policy28 or InsDoc28, then the process is suspended for 30 minutes.

XML request

```
<SuspendProcessRequest xmlns=
"http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>cmdb21s</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>o8rmond</Password></LoginData>
  </AuthenticationData>
  <WorkPackageURI>89 3 ICM7 cmdb21s11 WORKPACKAGE58 26
  A1001001A04H17B32803I2340818 A04H17B33113A150611 03
  204</WorkPackageURI>
  <Duration>30</Duration>
  <ResumeListDefinition>
    <RequiredItemTypeName>Policy28</RequiredItemTypeName>
    <QuantityNeeded>5</QuantityNeeded>
  </ResumeListDefinition>
  <ResumeListDefinition>
    <RequiredItemTypeName>InsDoc28</RequiredItemTypeName>
    <QuantityNeeded>25</QuantityNeeded>
  </ResumeListDefinition>
</SuspendProcessRequest>
```

As a result, DB2 Content Manager returns an XML SuspendProcessReply that indicates success.

Resuming a process with ResumeProcessRequest

You can force a suspended process to resume by specifying the associated work package in a ResumeProcessRequest. This checks the work package item back in, and resets the work package's SuspendState attribute to false. No routing or checkout of the associated work item is performed.

To resume DB2 Content Manager document routing process, create a ResumeProcessRequest that identifies the following information:

```
<ResumeProcessRequest>
  <AuthenticationData> ... </AuthenticationData>
  <WorkPackageURI> ... <WorkPackageURI>
</ResumeProcessRequest>
```

<ResumeProcessRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential. For more information about authentication data, see "Authenticating Web service requests for security" on page 465.

<WorkPackageURI> (required)

Specifies the persistent identifier of the work package in the process that you want to resume.

The following example query resumes a process for the specified work package.

XML request

```
<ResumeProcessRequest xmlns=
"http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>cmdb21s</ServerName></ServerDef><LoginData>
    <UserID>icmadmin</UserID><Password>o8rmond</Password></LoginData>
  </AuthenticationData>
  <WorkPackageURI>89 3 ICM7 cmdb21s11 WORKPACKAGE58 26
  A1001001A04H17B32803I2340818 A04H17B33118B569521 03
  204</WorkPackageURI>
</ResumeProcessRequest>
```

As a result, DB2 Content Manager returns an XML ResumeProcessReply that indicates success.

Batching multiple requests in XML requests

You can combine multiple requests in a single message using a BatchRequest wrapper tag. A batch request sequentially runs each request inside of it on the same connection. It can improve performance by decreasing the amount of network traffic required for multiple requests.

Batch requests use a single set of authentication data for all sub-requests (defined at the batch request level). Any sub-requests nested in the batch request run on the connection obtained using that authentication data.

If you nest sub-requests inside of transaction tags, then all of the sub-requests roll back if one of them fails.

To batch multiple requests in one message, create a BatchRequest that identifies the following information (text in brackets are optional):

```
<BatchRequest>
  <AuthenticationData> ... </AuthenticationData>
  [ <!-- Insert any number of requests here --> ]
  [ <Transaction><!-- Insert any number of requests here --></Transaction> ]
</BatchRequest>
```

<BatchRequest> elements:

<AuthenticationData> (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and

password (Password)--or a WebSphere SSO credential. For more information about authentication data, see “Authenticating Web service requests for security” on page 465.

<Transaction> (optional)

Identifies which requests to undo during a failure. If any request within the <Transaction> tag fails, then all requests in the same tag rollback. You cannot nest <Transaction> tags.

The following example deletes one policy and one claim in the same BatchRequest:

XML request

```
<BatchRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
    <ServerDef><ServerType>ICM</ServerType>
    <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
      <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
    </AuthenticationData>
    <Transaction>
      <Requests>

        <DeleteItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
          <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
            ...
          </AuthenticationData>
          <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B44211H0825818
A04I10B44211H082581 14 1029&server=icmnlsdb&dsType=ICM" />
        </DeleteItemRequest>

        <DeleteItemRequest
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
          <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
            ...
          </AuthenticationData>
          <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
3 ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B44213F1658218
A04I10B44213F165821 14 1027&server=icmnlsdb&dsType=ICM" />
        </DeleteItemRequest>

      </Requests>
    </Transaction>
  </BatchRequest>
```

As a result, DB2 Content Manager returns an XML BatchReply that wraps the replies from every XML request that you batched.

XML reply

```
<BatchReply
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <RequestStatus success="true"></RequestStatus>
  <Transaction>
  <Replies>

  <DeleteItemReply
    xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
    <RequestStatus success="true"></RequestStatus>
  </DeleteItemReply>
  <DeleteItemReply
    xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
    <RequestStatus success="true"></RequestStatus></DeleteItemReply>

  </Replies>
  </Transaction>
</BatchReply>
```

Chapter 12. Working with the Web services

DB2 Content Manager provides a self-contained, self-describing modular interface, called the Web services interface, that you can use within your applications, with other Web services interfaces, or in complex business processes to seamlessly perform actions against a DB2 Content Manager system. A Web service interface is a reusable, loosely coupled, software component that can be located, published and invoked through a network, like the Web.

The Web services interface allows you to dynamically integrate your applications with DB2 Content Manager, regardless of the programming language they were written in and the platform they reside in. You can use the Web services interface to do something as simple as view a text document or you can incorporate the Web services interface into more complex business applications or processes.

For example, in an insurance scenario, you can incorporate a Web services interfaces into an existing Web application that allows your customers to print their current auto policy. Furthermore, you can incorporate another Web services interface into the same application that allows your customer to view the current Blue Book value of their car.

This section provides the following information about the Web services interface offered by DB2 Content Manager:

- “Web services overview”
- “Understanding the DB2 Content Manager Web services implementation” on page 515
- “Integrating basic Web services into your applications or processes” on page 517

Web services overview

Web services is an emerging technology that is becoming the technology of choice for application integration. The key attribute of Web services is that they define a program-to-program, services-oriented communications model that is based on an XML messaging format. The Web services model is built on existing and emerging standards, such as Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Hyper Text Transfer Protocol (HTTP), and the Web Services Description Language (WSDL).

XML is an extensible tag language that can describe complicated structures in ways that are easy for programs, and people, to understand. Web services depend heavily on XML. XML uses textual data instead of binary data to represent things, like integers, that are often represented differently by the various hardware and software programming languages being used today. The textual way that XML represents data makes it language and platform independent. This independence save you time and resource when integrating your applications with DB2 Content Manager.

Simple Access Object Protocol (SOAP) is an XML-based messaging protocol that is used as the basis for Web services based interactions between two applications. All Web services communication is done by using SOAP messages. A SOAP message contains the following elements:

- Envelope

- Header (optional)
- Body
- Attachments (optional)

Typically, a SOAP envelope, with zero or more attachments, represents a SOAP message. The SOAP message envelope contains the header and the body of the message. The SOAP message attachments enable the message to contain data, which can include XML and non-XML data (like text and binary files). SOAP headers are used to describe the context and the purpose of the message. SOAP headers also provide mechanisms to extend a SOAP message for adding features and defining high-level functionality such as security, priority, and auditing.

SOAP allows you to invoke Web services in two ways: RPC (Remote Procedure Call) messaging and document style messaging. The DB2 Content Manager Web services use the document style method for invoking Web services because it is much more flexible than the RPC method.

In a service oriented architecture, the interface definition is crucial. It is the interface definition that serves as the contract between what the Web service provides and what the client can expect. Web services use WSDL, another set of XML tags that are used to describe the Web service interface. The types of things that WSDL describes are the location of the Web service, how to connect to it, which parameters must be passed in the SOAP request, and which return values should come back. The WSDL also provides binding information.

The Web services model leverages the XML, HTTP, SOAP, and WSDL technologies and protocols to provide an environment that makes application integration easier, faster, and more cost effective. Web services allow any network-enabled, XML-aware application to invoke a Web service regardless of the programming language or operating system involved.

Web services provide the following advantages:

Flexibility

Universal interfaces do not have to worry about the inevitable changes in software caused by changing business needs.

Agility and productivity

Rapid application assembly tools allow you to quickly integrate Web services into new business processes or experiment with new business ideas.

Cost savings

Reduce staffing requirements, replace paper processing, reduce errors.

Leverage existing investments

You can use old software in new ways by building a Web services layer for universal access.

In order to work with the DB2 Content Manager Web services, you must have a working knowledge about them. To find out more about the Web services standards, see the World Wide Web Consortium (W3C) Web site: <http://www.w3.org>.

Understanding the DB2 Content Manager Web services implementation

The DB2 Content Manager Web services interface architecture is based on a messaging communications model, in which whole documents are exchanged between service clients and servers. This messaging based model provides several benefits.

One benefit of the document messaging based model is that the XML specification was developed to allow ordinary data, that is usually locked up in a proprietary format, to be described in an open format that is human readable, self-describing, and self-validating. When a Web service uses document messaging, it can use the full capabilities of XML to describe and validate a high-level business document. Another benefit is that even though enhancements and changes are made to the XML schema, the calling application will not break.

Lastly, the document messaging model makes object exchange more flexible, because the design of a business document is often well suited to object-oriented architectures. As a result, two applications can be designed to exchange the state of an object by using XML. In contrast with object serialization, in an object exchange each end of the exchange is free to design the object as it sees fit as long as the exchange conforms to the agreed upon XML document format. One reason for not using object serialization is to support client-side and server-side implementations of an object. Many current industry-specific XML schemas are designed as client-server architectures in which the processing that is done at the client is separate from the processing intended at the server. As is often the case, the client is simply requesting or saving information in a specific document format that is persisted at the server.

The main components in the DB2 Content Manager Web services model include the requester, the Web services server, the XML beans layer, and the DB2 Content Manager repository. They interact in the following steps:

1. A requester makes a call to the Web services server.
2. The DB2 Content Manager Web services server analyzes and extracts the XML message from the SOAP envelope.
3. The XML message is sent to the DB2 Content Manager XML beans layer.
4. The XML beans transform the XML into multiple calls to the underlying DB2 Content Manager APIs.
5. The APIs access the data in the repository and return values to the XML beans.
6. The return values from the APIs are transformed into an XML response message by the XML bean. This message contains the request status, response data and attachments, and exception information, if applicable.
7. The message is returned to the DB2 Content Manager Web services server.
8. The Web services server creates a SOAP message, which can include attachments, from the response data and returns the message to the requester.

Figure 36 on page 516 depicts the steps involved in document processing using Web services. A SOAP request is sent to the Web services server to store an insurance claim. The Web server processes the SOAP request and sends the data onto the server. The claim is stored into the library server and pictures associated with the claim are stored in the resource manager.

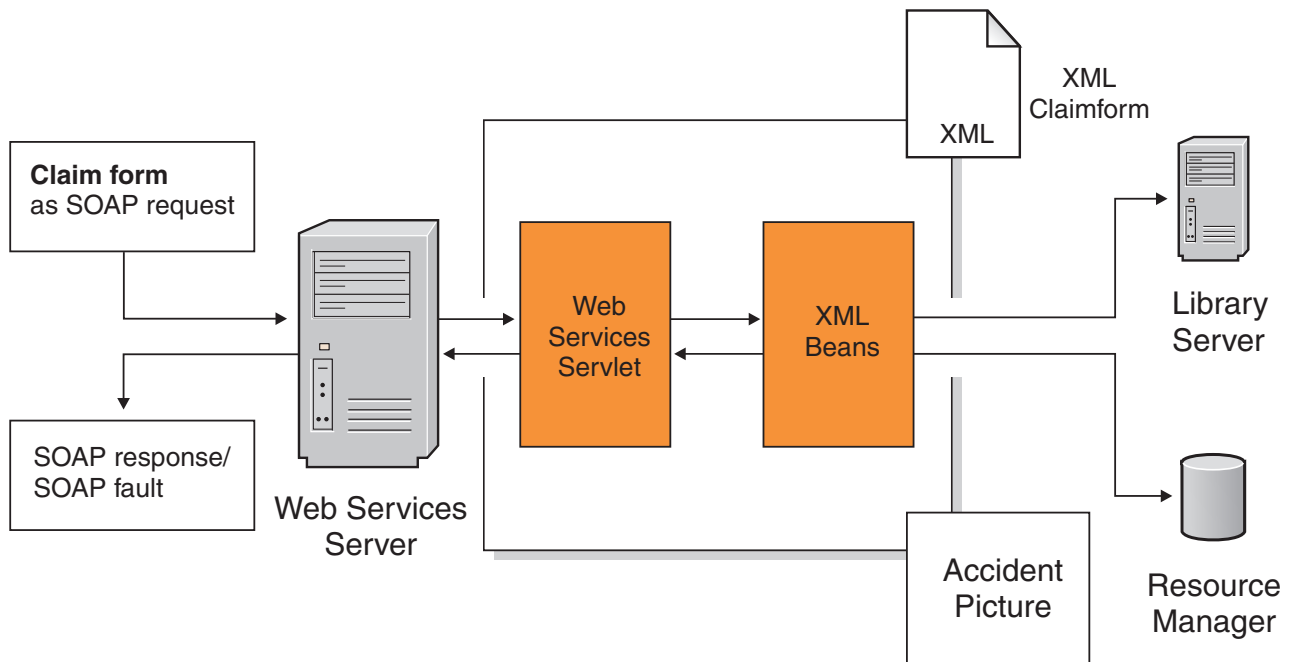


Figure 36. Processing a document using Web services

Working with the Web service in development tools

The DB2 Content Manager Web service is a messaging-based communication model that defines loosely coupled and document-driven communication. The client service requester invokes the Web service by sending it a complete XML document that represents a particular request for a DB2 Content Manager operation, such as search. The DB2 Content Manager Web service provider receives the document, processes it, and returns a message, as an XML document.

When you install of the DB2 Content Manager Web service, two WSDL locations that describe the operations and end points of the Web service are provided by DB2 Content Manager. Your application environment is the determining factor for choosing which WSDL location to use.

There are a number of Web services toolkits that can take a WSDL file and create a set of classes for client-side representation of the Web service, the request, and reply messages. The benefit of using development toolkits is that you do not have to create the XML document yourself because toolkits can create classes that generate the XML requests for you. The toolkit serializes the classes into XML and creates and exchanges the SOAP messages with the Web service. This makes client-side development much easier and faster.

In order for a tool to create the classes, the WSDL must thoroughly describe the syntax of the input and output messages, as well as the operations. Since the schema of the user defined item types are not necessarily known at installation time, you must create the WSDL for the item types after you have completely installed DB2 Content Manager. You can generate a WSDL for any item type using the DB2 Content Manager system administration client, and use that WSDL to perform operations provided by the Web service.

Tools such as the WSDL.exe provided by the Microsoft .NET Framework SDK or the Web Reference feature in Microsoft Visual Studio .NET can take a WSDL file and create a set of classes that you can use to invoke the DB2 Content Manager Web service. The WSDLs generated by the DB2 Content Manager system administration client support clients built in .NET and in Java. Figure 37 illustrates the process of creating WSDLs for use in a .NET environment.

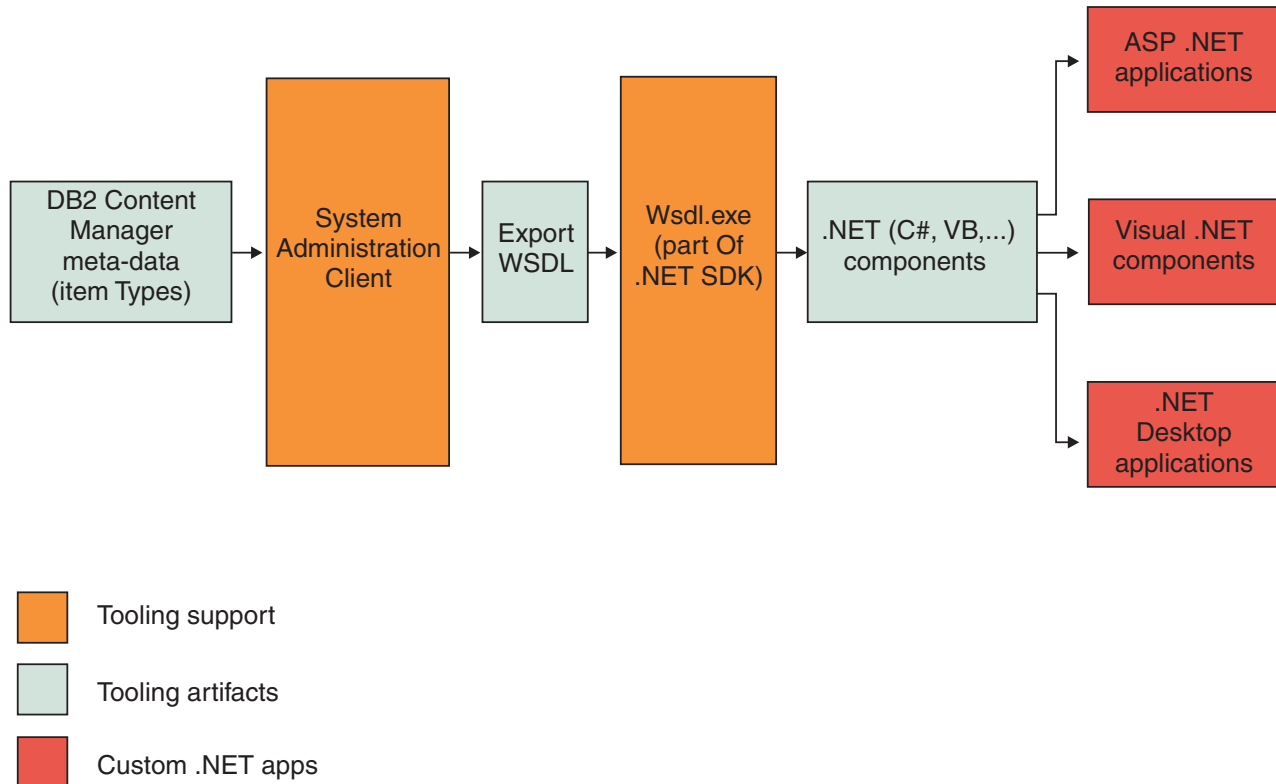


Figure 37. Tooling support for use in a .NET environment

Integrating basic Web services into your applications or processes

This section explains how to develop client applications in order to interact with the DB2 Content Manager Version 8 Release 3 Web services interface. You can communicate with the interface through the Web services Description Language (WSDL) in the following two ways:

WSDL generation

You can write an application that uses a WSDL utility to automatically handle the XML/SOAP requests and responses based on the structure of your item types.

The sample classes for this application are written in C#. This requires the Web service support and `wsdl.exe` utility provided in Microsoft Visual Studio .NET 2003. All C# sample classes are located in `IBMCMROOT/samples/webservices/CMWebServiceClient`.

For details, see “Getting started with the Web services in a .NET environment” on page 518.

XML/SOAP requests

You can write an application to send your own XML requests through a

SOAP envelope to the Web services server. This server translates the request into calls on the XML Handler bean, and then sends XML/SOAP responses back to your application.

The sample classes for this application are written in Java, which require a Web services JAR file from the WebSphere Studio Version 5.1 or Version 6.0 Web services toolkit. All sample Java classes are located in *IBMCMROOT/samples/webservices/GenericWebServiceSample*.

For details, see “Getting started with the Web services in a Java environment” on page 520.

WebSphere Studio Application Developer (WSAD) only: If you need are creating a Web services client using WebSphere Studio Application Developer (WSAD) Version 5.1, then within that wizard you must select **Define custom mapping for namespace to package**, and specify a different package for each of the namespaces:

- `http://www.ibm.com/xmlns/db2/cm/api/1.0/schema`
- `http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema`
- `http://www.ibm.com/xmlns/db2/cm/webservices/1.0/schema`

and a package for no namespace, which you can specify as an empty string.

Business Process Choreographer only: If you using the generated wsdl from the System Admin application for Business Process Choreographer, you must edit the `cmbmessages_modified.xsd` file (which is located in the ZIP file along the WSDL file), and change the line:

```
<xs:import xmlns="http://www.w3.org/2001/XMLSchema"
schemaLocation="itemtype_modified.xsd"/> to <xs:include
xmlns="http://www.w3.org/2001/XMLSchema"
schemaLocation="itemtype_modified.xsd"/>.
```

Getting started with the Web services in a .NET environment

For .NET clients, the WSDL does not describe the syntax of the input and output of the messages. They are defined as `xs:AnyType`. You should use the XML Beans messages schema file (`cmbmessages.xsd`) and the item type schema files to generate the XML request documents and send them to the Web service using the URL specified in the WSDL file. For details about `cmbmessage.xsd`, see “Programming runtime operations through the XML JavaBeans” on page 461

Toolkits such as the Microsoft SOAP Toolkit, provide low-level APIs for generating and exchanging SOAP messages. These APIs allow you to specify an XML document that represents the body of a SOAP message and will send the document to the Web service URL and return the reply document as part of the SOAP message. This is a low level interaction with the web service. This type of interaction allows the most flexibility since the Web service interface does not change, even if the XML schema changes.

The disadvantage to using the Web service in this manner is that there is the burden, on the development side, of generating XML messages and dealing with low level APIs for sending and receiving SOAP messages. In this case, the WSDL is only used for specifying the end point URL of the Web service.

To create XML/SOAP requests, you can write a `CMWebServiceClient` application that utilizes a WSDL utility. A WSDL utility can automatically process your

XML/SOAP requests and responses by representing them as proxy classes. For example, Microsoft provides a `wsdl.exe` utility that can represent XML documents as C# proxy classes.

To write an application that interfaces with Microsoft .NET's WSDL utility, you would perform the following steps:

1. Install of the following software:
 - DB2 Content Manager Version 8 Release 3 Web services toolkit
 - Microsoft Visual Studio .NET 2003
 - .NET Framework SDK Version 1.1 from <http://www.microsoft.com/downloads/details.aspx?FamilyID=9b3a2ca6-3647-4070-9f41-a333c6b9181d&DisplayLang=en>
 - .NET Web Service Enhancements SP1 from <http://www.microsoft.com/downloads/details.aspx?familyid=06255a94-2635-4d29-a90c-28b282993a41&displaylang=en>
2. Load the First Steps XYZ insurance samples.
3. In the `%IBMCMROOT%\samples\webservices\CMWebServiceClient\CMWebService.cs` file, replace all instances of `localhost` with the name of your Web services server.
4. Load the `CMWebServiceClient.csproj` into Visual Studio .Net.
5. Program the Web services application using `CMWebServiceClient.cs` as guidance. For details about programming Web services requests in C#, see "Programming Web services requests in a .NET environment."
6. Run the sample by entering the command:
`CMWebServiceClient.exe icmnlsdb icmadmin password`

where *icmnlsdb* represents the DB2 Content Manager server, *icmadmin* represents your system administration ID, and *password* represents the password.

Programming Web services requests in a .NET environment

After setting up your Java Web services environment in "Getting started with the Web services in a Java environment" on page 520, you can use `CMWebServiceClient.cs` to walk you through a sample of creating a Web services request. In summary, a Java application would contain the following code:

Creating a Web service object

To instantiate a Web services object, you can use the following example:

```
CMWebService webservice = null;  
webservice = new CMWebService();  
webservice.Timeout = 60000;
```

Authenticating the Web services request

For security, you must create an authentication object in each request. For instructions, see "Authenticating Web services requests for security" on page 523.

Creating a new instance of an item (if applicable)

The `CMWebServiceClient.cs` sample creates a new instance of an `XYZ_InsPolicy` item. For details, see "Creating a new instance of an item through Web services" on page 524.

Wrapping the XML request

The `CMWebServiceClient.cs` sample specifies direct XML requests through the WSDL generation utility. For instance:

```

elementRequest request = new RetrieveItemRequest();
request.AuthenticationData = authData;
request.attribute = elementattribute.att_value;
elementReply reply=webservice.element(request);

```

For exact details on the XML request syntax, see “Programming runtime operations through the XML JavaBeans” on page 461.

Attaching binary content (if applicable)

You can use one of two standard binary message formats to attach content parts to a request. For instructions, see “Attaching binary content parts to items in Web services” on page 526

The following example wraps an XML request to retrieve the XYZ insurance policy (and a URL for its resource part) through its persistent identifier.

C# sample

```

public XYZ_ClaimForm retrieveClaimWithResourceURL(
    AuthenticationData authData, string pidURI) {
    RetrieveItemRequest request = new RetrieveItemRequest();
    request.AuthenticationData = authData;
    request.retrieveOption =
        RetrieveItemRequestRetrieveOption.CONTENT_WITH_LINKS;
    request.contentOption = RetrieveItemRequestContentOption.URL;
    request.Item = new RetrieveItemRequestItem();
    request.Item.URI = pidURI;
    RetrieveItemReply reply = webservice.RetrieveItem(request);
    if (reply.RequestStatus.success == true) {
        return reply.Item.ItemXML.XYZ_ClaimForm; }
    else {
        Console.WriteLine("Retrieve Policy failed.");
        displayErrorInfo(reply.RequestStatus.ErrorData);
        return null; }
}

```

Getting started with the Web services in a Java environment

You can customize your own Java client to create the XML requests to send to Web services. For Java clients, there is one operation called `processXMLRequest`, which describes two input parameters. The first parameter is an XML string that represents the XML request for the web service. The second parameter is a `javax.mail.internet.MimeMultipart` object which represents the attachment representing a document or resource object. You must generate this XML string using the XML Beans messages schema file (`cmbmessage.xsd`) and the item type schema files. For details about `cmbmessage.xsd`, see “Programming runtime operations through the XML JavaBeans” on page 461.

You can use any JAX-RPC based client toolkit to generate the classes that will invoke the web service and pass the parameters back and forth to the web service. WebSphere Version 5.1 provides a client- side tool called `WSDL2Java` that you can use to generate the client- side classes for the Web service. Since the WSDL file does not define the syntax of the XML documents, the interface of the Web service does not change if the XML schema for the request changes.

To write an application in the Java environment, you would perform the following steps:

1. Install of the following software:
 - DB2 Content Manager Version 8 Release 3 Web services toolkit

- WebSphere Application Server Version 5.1
2. Load the First Steps XYZ insurance samples.
 3. In the *IBMCMROOT/samples/webservices/GenericWebServiceSample/sample/CMBGenericWebServiceServiceLocator.javafile*, modify the *CMBGenericWebService_address* variable with the name of your Web services server rather than *localhost*.
 4. Program the Web services application using *GenericWebServiceSample.java* as guidance. For details about programming Web services requests in Java, see “Programming Web services requests in a Java environment.”
 5. Compile the proxy classes and *GenericWebServiceSample.java* with the CLASSPATH with the following WebSphere JAR files: *activation.jar*, *j2ee.jar*, *mail.jar*, *qname.jar*, *webservices.jar*, and *wsdl4j.jar*.
 6. Run the sample by entering the command:

```
java GenericWebServiceSample icmnlbdb icmadmin password
```

where *icmnlbdb* represents the DB2 Content Manager server, *icmadmin* represents your system administration ID, and *password* represents the password.

Programming Web services requests in a Java environment

After setting up your Java Web services environment in “Getting started with the Web services in a Java environment” on page 520, you can use *GenericWebServiceSample.java* to walk you through a sample of creating a Web services request. In summary, a Java application would contain the following code:

Creating a Web service object

To instantiate a Web services object, you can use the following example:

```
CMBGenericWebServiceService cs =
    new CMBGenericWebServiceServiceLocator();
cmbService = cs.getCMBGenericWebService();
```

Authenticating the Web services request

For security, you must create an authentication object in each request. For instructions, see “Authenticating Web services requests for security” on page 523.

Creating a DOM out of an XML string

Building a DOM document out of an XML message string helps you read the elements inside of the XML replies.

```
if (factory == null) {
    factory = DocumentBuilderFactory.newInstance();
}
builder = factory.newDocumentBuilder();
Document document = null;
document = builder.parse(new InputSource(new StringReader(replyXML)));
```

Creating a new instance of an item (if applicable)

The *GenericWebServiceSample.java* sample creates a new instance of an *XYZ_InsPolicy* item. For details, see “Creating a new instance of an item through Web services” on page 524.

Wrapping the XML request

The *GenericWebServiceSample.java* sample passes parameters into pre-defined message templates to create XML requests. For instance:

```
String requestXML = MessageFormat.format(
    SampleMessageTemplate.TEMPLATE,
    new Object[] { authenticationDataXML, pid });
```

```

CMBXMLResponse response = null;
response = cmbservice.processXMLRequest(requestXML, null);
String replyXML = response.getXmlResponseText();
return replyXML;

```

All of the Web services message templates are defined in `SampleMessageTemplate.java`. The basic operations templates include:

- AUTHENTICATION_DATA_TEMPLATE
- CREATE_ITEM_TEMPLATE
- QUERY_TEMPLATE
- RETRIEVE_ITEM_WITH_RESOURCE_URL_TEMPLATE
- RETRIEVE_ITEM_WITH_ATTACHMENTS_TEMPLATE
- UPDATE_CLAIM_TEMPLATE
- CREATE_LINKS_TEMPLATE
- DELETE_LINKS_TEMPLATE
- UPDATE_POLICY_TEMPLATE
- UPDATE_POLICY_EXT_TEMPLATE
- DELETE_ITEM_TEMPLATE
- BATCH_DELETE_TEMPLATE
- ADD_TO_FOLDER

The XML item templates include:

- XYZ_InsPolicy_TEMPLATE
- XYZ_InsPolicy_FOLDER_TEMPLATE
- XYZ_InsPolicy__XYZ_Insured_TEMPLATE
- XYZ_InsPolicy__XYZ_VIN_TEMPLATE
- ICM_BASE_TEMPLATE
- XYZ_ClaimForm_TEMPLATE

The document routing templates include:

- LIST_PROCESS_TEMPLATE
- START_PROCESS_TEMPLATE
- LIST_WORKPACKAGES_TEMPLATE
- CONTINUE_PROCESS_TEMPLATE
- TERMINATE_PROCESS_TEMPLATE

For exact details on the XML request syntax, see “Programming runtime operations through the XML JavaBeans” on page 461.

Attaching binary content (if applicable)

You can use one of two standard binary message formats to attach content parts to a request. For instructions, see “Attaching binary content parts to items in Web services” on page 526

Parsing the Web services request

The `GenericWebServiceSample.java` sample contains various methods for validating XML requests and reporting errors.

The following example wraps an XML request to retrieve the XYZ insurance policy (and a URL for its resource part) through its persistent identifier.

Java sample

```
public CMDocument retrievePolicyWithResourceURL(
String authenticationDataXML, String pid) {
String requestXML = MessageFormat.format(
    SampleMessageTemplate.RETRIEVE_ITEM_WITH_RESOURCE_URL_TEMPLATE,
    new Object[] { authenticationDataXML, pid });
CMBXMLResponse response = null;
// call the web service with the xml request message
try {
response = cmbservice.processXMLRequest(requestXML, null);
} catch (RemoteException e) {
e.printStackTrace();
return null;
}
// Get the DOM object representing the xml message
Document document = getDocument(response.getXmlResponseText());
if (document == null) {
return null;
}
// parse the status of the response from web service
if (parseRequestStatus(document) != true) {
return null;
}
// return the array of PIDs for the resources associated
// with the retrieved document
Element policyElement = getElement(document, "XYZ_InsPolicy");
if (policyElement == null) {
return null;
}
CMDocument policy = parsePolicy(policyElement);
return policy;
}
```

Authenticating Web services requests for security

Every time you make a request to the Web services, you must pass in a DB2 Content Manager username and password, or a WebSphere credential token associated with a DB2 Content Manager user. If a user does not have the privilege to perform the specific request, then the request is not processed and an error is returned in the SOAP reply. For example, if a user wants to make change to an insurance policy, but only has view privileges, the user cannot make any changes to the policy.

Important: By default, the username and password passed in the Web services request are not encrypted. This is not a big issue if all of the Web service requests are being processed within the firewall. However, if the client is outside the firewall, and you should use SSL to send your SOAP requests.

To authenticate your Web service requests, create an `AuthenticationData` object. You must then include this object in every request.

The following specific C# example returns an initialized `AuthenticationData` object where *dstype* represents the data source type (for example, ICM), *server* represents the DB2 Content Manager hostname (for example, cmdb21s), *username* represents your DB2 Content Manager user ID (for example, icmadmin), and *password* represents your log in password.

C# sample

```
public AuthenticationData setupAuthenticationData(string server, string
userid, string password)
{
    AuthenticationData authData = new AuthenticationData();
    ServerDef serverDef = new ServerDef();
    serverDef.ServerName = server;
    authData.ServerDef = serverDef;
    AuthenticationDataLoginData loginData = new
        AuthenticationDataLoginData();
    loginData.UserID = userid;
    loginData.Password = password;
    authData.LoginData = loginData;
    return authData;
}
```

The following generic Java sample creates an authentication object where *dstype* represents the data source type (for example, ICM), *server* represents the DB2 Content Manager hostname (for example, cmdb21s), *username* represents your DB2 Content Manager user ID (for example, icmadmin), and *password* represents your log in password.

Java sample

```
public String createAuthenticationDataXML( String dstype,
String server, String username, String password) {
    return MessageFormat.format(
        SampleMessageTemplate.AUTHENTICATION_DATA_TEMPLATE,
        new Object[] { dstype, server, username, password });
}
```

Creating a new instance of an item through Web services

For information about how to create an item in DB2 Content Manager, see “Creating an item” on page 129. The following examples create a new instance of an XYZ_InsPolicy item called setupPolicy.

C# sample

```
private XYZ_InsPolicy setupPolicy(string[,] insured,
string street, string city, string state, string zip,
string policyNumber, string[] vins,string[,] resources) {
XYZ_InsPolicy policy = new XYZ_InsPolicy();
policy.XYZ_City = city;
policy.XYZ_Street = street;
policy.XYZ_State = state;
policy.XYZ_ZIPCode = zip;
policy.XYZ_PolicyNum = policyNumber;
policy.XYZ_VIN = new XYZ_InsPolicyXYZ_VIN[vins.Length];
for (int i=0; i<vins.Length; i++) {
policy.XYZ_VIN[i] = new XYZ_InsPolicyXYZ_VIN();
policy.XYZ_VIN[i].XYZ_VIN = vins[i]; }
policy.XYZ_Insured =
new XYZ_InsPolicyXYZ_Insured[insured.GetLength(0)];
for (int i=0; i<insured.GetLength(0); i++) {
policy.XYZ_Insured[i] = new XYZ_InsPolicyXYZ_Insured();
policy.XYZ_Insured[i].XYZ_InsrdfName = insured[i, 0];
policy.XYZ_Insured[i].XYZ_InsrdfLName = insured[i, 1]; }
// Document parts of the policy (ICM base parts only)
policy.ICMBASE = new ICMBASE[resources.GetLength(0)];
for (int i=0; i<resources.GetLength(0); i++) {
policy.ICMBASE[i] = new ICMBASE();
policy.ICMBASE[i].resourceObject = new LobObjectType();
policy.ICMBASE[i].resourceObject.label=new LobObjectTypeLabel();
policy.ICMBASE[i].resourceObject.label.name = resources[i, 0];
policy.ICMBASE[i].resourceObject.MIMEType = resources[i, 1]; }
return policy;
}
```

Java sample

```
public String generatePolicyDataXML(String[] policyInfo,
String[] insured, String[] vins, String[][] resources) {
StringBuffer insuredXML = new StringBuffer();
StringBuffer vinsXML = new StringBuffer();
StringBuffer resourcesXML = new StringBuffer();
for (int i = 0; i < insured.length; i++)
insuredXML.append(
MessageFormat.format(
SampleMessageTemplate.XYZ_InsPolicy__XYZ_Insured_TEMPLATE,
new Object[] { insured[i][0], insured[i][1] }));
for (int i = 0; i < vins.length; i++)
vinsXML.append(
MessageFormat.format(
SampleMessageTemplate.XYZ_InsPolicy__XYZ_VIN_TEMPLATE,
new Object[] { vins[i] }));
if (resources != null) {
for (int i = 0; i < resources.length; i++)
resourcesXML.append(
MessageFormat.format( SampleMessageTemplate.ICM_BASE_TEMPLATE,
new Object[] { resources[i][0], resources[i][1] }));
} else {
resourcesXML.append("");
}
return MessageFormat.format(
SampleMessageTemplate.XYZ_InsPolicy_TEMPLATE,
new Object[] {policyInfo[0],policyInfo[1],policyInfo[2],
policyInfo[3],policyInfo[4],insuredXML.toString(),
vinsXML.toString(),resourcesXML.toString()});
}
```

Attaching binary content parts to items in Web services

The DB2 Content Manager Web services samples rely on a couple of standard binary message formats to send attachments (such as base parts, annotations, or notelogs) inside XML requests:

Multipurpose Internet Mail Extensions (MIME)

Standard messaging protocol that can identify binary files and types though the Content-ID or Content-Location headers in a SOAP envelope. XML items in DB2 Content Manager use the `MIMETYPE` attribute in the `<resourceObject>` element to define the encoding. For example:

```
<ICMBASE>
  <resourceObject MIMETYPE="image/jpeg"
    xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema">
    <label name="image 1" />
  </resourceObject>
</ICMBASE>
```

The Java Web services samples use MIME types to attach resource content parts.

The following example creates a `MimeMultipart` object to hold content parts (in the *resources* array) within the DB2 Content Manager items.

Java sample

```
public MimeMultipart setupAttachments(String[] [] resourceData) {
    MimeMultipart mp = new MimeMultipart();
    for (int i = 0; i < resourceData.length; i++) {
        MimeBodyPart mbp = new MimeBodyPart();
        DataHandler handle =
            new DataHandler(new FileDataSource(resourceData[i][2]));
        try {
            mbp.setDataHandler(handle);
            // we need to create a header called ID which matches the label value
            // of the resourceObject specified in the policy xml on the server
            // this ID value is used to match the appropriate resourceObject
            mbp.addHeader("ID", resourceData[i][1]);
            mbp.addHeader("MimeType", resourceData[i][0]);
        } catch (MessagingException e) {
            System.out.println(
                "Failed in creating a MimeBodyPart for attachments");
            e.printStackTrace();
            return null;
        }
        try {
            mp.addBodyPart(mbp);
        } catch (MessagingException e1) {
            System.out.println(
                "Failed in creating a MimeMultiPart for attachments");
            e1.printStackTrace();
            return null;
        }
    }
    return mp;
}
```

Direct Internet Message Encapsulation (DIME)

DIME is a Microsoft-defined protocol that can specify the length, encoding, and payload of a file ahead of time in the header fields to save the time of calculating it. This protocol is best for handling digitally signed or large binary files. The C# Web services samples use DIME to attach resource content parts.

The following example creates a DIME attachment object to hold content parts (in the *resources* array) within the DB2 Content Manager items.

C# sample

```
DimeAttachment[] setupAttachments(string[,] resources) {
    DimeAttachment[] attachments =
        new DimeAttachment[resources.GetLength(0)];
    for (int i=0;i<resources.GetLength(0);i++) {
        // resources[i,0] is the ID of the attachment. This should match
        // the label of the document part on the server side, the ID of the
        // attachment and the label of the document part are used to match
        // the attachment with its associated document part
        // resources[i,1] is the mime type of the document part
        // resources[i,2] is the file path to the document part
        attachments[i] = new DimeAttachment(resources[i,0],
            resources[i,1], Microsoft.Web.Services.Dime.TypeFormatEnum.MediaType,
            resources[i,2]);
    }
    return attachments;
}
```

Chapter 13. Working with the Java document viewer toolkit

The document viewer toolkit is a set of Java APIs that facilitate displaying and annotating documents. The toolkit includes a Swing based graphical class and class objects that provide document conversion and rendering capabilities. You can customize the existing graphical interface provided by the toolkit or you can build your own graphical interface.

Using the Java viewer toolkit, you can create a custom document viewer that you can use to access and annotate documents contained in your content servers. You can also create custom viewer applets and standalone applications that integrate with DB2 Content Manager. The DB2 Content Manager eClient uses the Java viewer toolkit to provide document viewing functionality. The eClient contains both an applet viewer and mid-tier conversion (HTML) viewing capability for TIFF and other document formats.

The Java viewer toolkit supports the following formats:

- TIFF, MODCA (containing IOCA, PTOCA), GIF, JPEG, PCX, Bitmap, and text documents that use pure Java
- Microsoft Office documents using a bridge to the Stellent INSO toolkit

The Java viewer toolkit provides the following capabilities:

- Multiple-document interface
- Thumbnail view of the pages of a document: You can click on a thumbnail view of a page to move to that page. The page displays surrounded by a red rectangle. You can drag or resize the rectangle to move to a different section in the page or to rescale the page.
- Multiple selection of pages
- Configurable toolbars
- Popup menus that you can customize
- Java Action objects for all viewer actions: You can use the Action objects to build a menu bar, because the viewer does not provide a menu bar.
- Rotating, zooming, inverting, and enhancing pages
- Page and document navigation: First, next, previous, last page, and jump to a page.
- Fit to width, height, and to window
- Printing
- Copying and pasting
- Creating and editing graphical annotations: The annotation graphics provided include a box, circle, line, arrow, text, highlight, sticky note, and pen.
- Page manipulation: See "Working with the page manipulation functions" on page 546 for a detailed description of the page manipulation capabilities.
- Page filtering: Enables you to display a subset of the pages of a document. This capability is provided through the programming interface only.
- Undo and redo for operations that modify the document, such as editing annotations and page manipulation
- Export dialog to write a document to the file system. The document can include annotations.

The Java viewer toolkit provides many GUI classes that you can use to build Swing-based applications. It also provides non-GUI classes that you can use for non Swing-based document viewing applications.

Figure 38 is an example of a generic document viewer that uses all of the default settings.

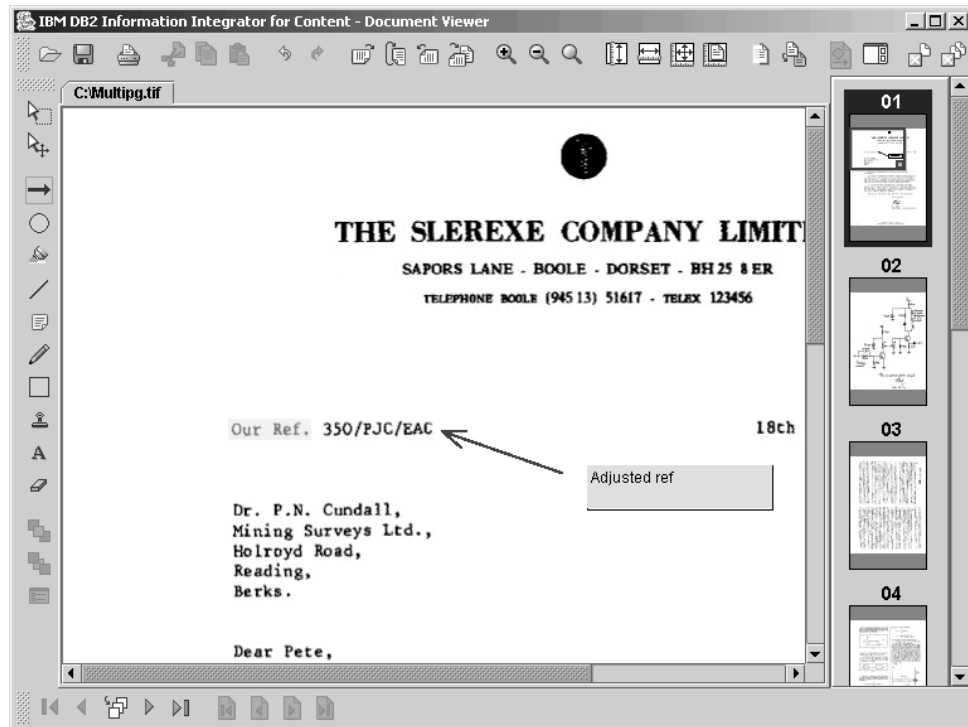


Figure 38. Generic document viewer

Viewer architecture

The Java viewer toolkit contains a Document Viewer and a Document Services bean. The beans provide a mechanism for integrating the viewer into applications based on Information Integrator for Content JavaBeans.

The toolkit also contains the Generic Doc Viewer, Streaming Doc Services, and Annotation Services classes. These 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 with the capability to manipulate the pages of a document. The engines provide parsed document images in various formats, such as TIFF and IOCA, as well as text and rich text documents, and office formats. You can also write additional document engines and plug them into the viewer toolkit architecture to support additional formats or alternative rendering for document formats.

The Annotation Services class enables you to manipulate annotations. The annotation engine only parses DB2 Content Manager specific annotation formats. However, you can write additional annotation engines to handle other annotation

storage formats. The following diagram shows the components that make up the Java Viewer Toolkit:

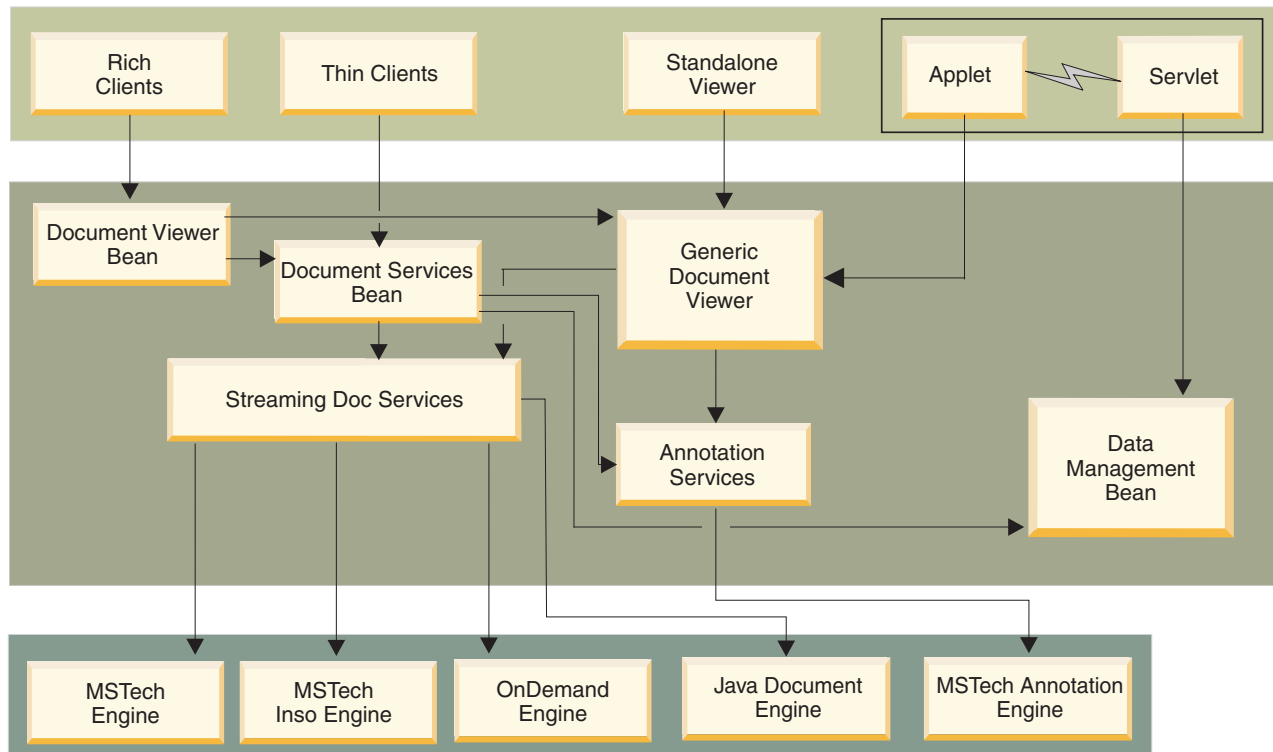


Figure 39. The components in the Java viewer toolkit

The document engines

Following is the list of document engines provided with the Java viewer toolkit:

- MS-Tech Document Engine (CMBMSTechDocumentEngine): This engine renders pages as images, and handles content types typically found on DB2 Content Manager. The document types supported by this engine include TIFF and MO:DCA™. This engine also supports GIF, JPEG, and plain text.
- MS-Tech INSO Engine (CMBMSTechInsoEngine): This engine supports Microsoft Office, Lotus SmartSuite®, and other office document formats.
- OnDemand Engine (CMBODDDocumentEngine). This engine converts line data to plain text, and combines AFP™ large object documents into a single AFP document. It also contains a bridge to the AFP2Web toolkit (available separately) to convert pages of AFP documents to HTML.
- Java Document Engine: This engine converts documents that are URL's to HTML with a forwarding link to the URL.

The engines listed above 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 to access the functionality provided by the document engines.

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 Document Engine and Java Document Engine are pure Java, and can be used on most operating systems. The other engines contain platform specific logic that restricts their use to the Windows platform.

The annotations engine

Information Integrator for Content provides the MS-Tech annotation engine to retrieve and update DB2 Content Manager annotations. The MS-Tech engine supports DB2 Content Manager Version 8.2 and Version 8.3, IBM Content Manager Version 8.1, Content Manager Version 7, and VI/400 annotations.

Example viewer architectures

To help you understand different ways that you can use the Java viewer toolkit, this section provides five example architectural designs that use the Java viewer toolkit and beans. Note that the examples in this section 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 a standalone viewer as a pallet on a Web page or for viewing documents that you obtained through e-mail. The figure below illustrates a standalone viewer architecture.

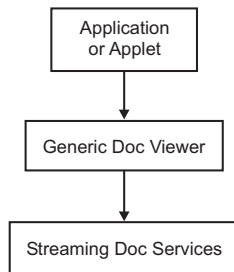


Figure 40. Standalone viewer

Java application

To create a production Java application use the Information Integrator for Content 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 that these viewers can have platform dependencies. Figure 41 illustrates the architecture you can use to build a Java application.

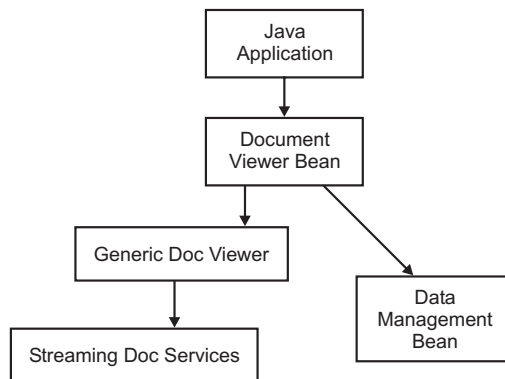


Figure 41. Java application

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 plug-in or native application to launch) to content types that are handled by the browser natively, such as HTML, GIF, JPEG, or for which plug-ins are readily available, such as PDF. Figure 42 illustrates a thin client architecture.

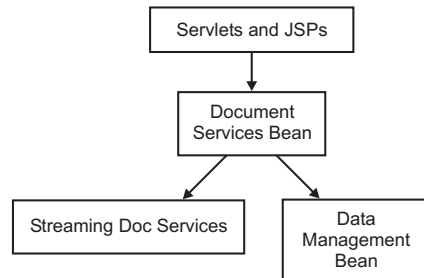


Figure 42. Thin client

Applet or servlet

A Web-based application can also provide document viewing and annotation editing capabilities using an applet or servlet approach. The eClient viewer applet uses this architecture. The applet can use the Generic Document Viewer to view the document. Some document types are stored in several parts on the content server to make sharing of common information, such as background forms, more efficient. The additional parts are requested by the Generic Document Viewer when needed. The applet containing the viewer satisfies the requests by sending HTTP requests to the servlet. On the servlet side, the content server using the CMBDataManagement bean obtains the requested information. Figure 43 illustrates an applet or servlet architecture.

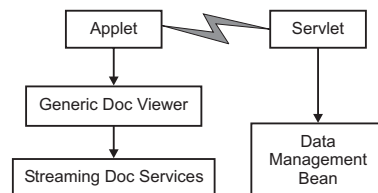


Figure 43. Applet or servlet

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 end 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.

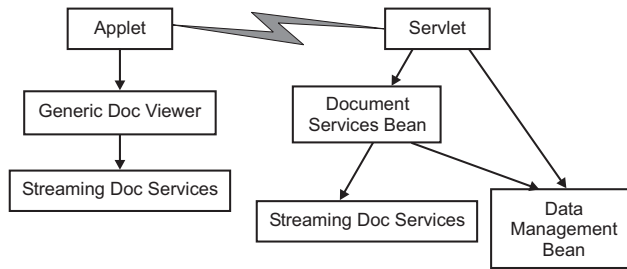


Figure 44. Dual-mode viewer

Creating a document viewer

To create a document viewer, you work primarily with the `CMBGenericDocViewer` class, which is the graphical user interface (GUI) for the document viewer. This GUI is based on the Java Foundation Class (JFC). By itself, the `CMBGenericDocViewer` is a GUI that relies on other classes to handle document rendering and annotation functions. These functions are provided by two non-visual classes, the document services class `CMBStreamingDocServices`, and the annotation services class `CMBAnnotationServices`.

Creating a standalone viewer application or applet

As you complete the following steps, see the Application Programming Reference located in the information center. Also see the standalone viewer application, `TGenericDocViewer.java`, and the viewer applet, `TViewerApplet.java`, samples located in the samples directory.

To begin, instantiate the `CMBGenericDocViewer` object. When instantiating the `CMBGenericDocViewer` object, you must create an instance of the `CMBStreamingDocServices` and `CMBAnnotationServices` class objects by implementing the following callback classes:

- `CMBStreamingDocServicesCallbacks` abstract class
- `CMBAnnotationServicesCallbacks` abstract class

The following code shows how to create the `CMBGenericDocViewer` object:

Creating the `CMBGenericDocViewer` object

```
// Create streaming doc services object
docServices = new CMBStreamingDocServices(
    new StreamingDocServicesCallbacks(), null);

// Create annotation services object
annoServices = new CMBAnnotationServices(
    new AnnotationServicesCallbacks());

// Create generic doc viewer object
genericDocViewer = new CMBGenericDocViewer(
    docServices, annoServices, configProps);

// Add generic doc viewer GUI to content pane
getContentPane().add(genericDocViewer, BorderLayout.CENTER);
```

1. Create a class that extends `CMBStreamingDocServicesCallbacks`. This class is used by `CMBGenericDocViewer` to retrieve and update document parts and resources. This includes methods to retrieve a background form for a document and the print privilege of a document. In the code sample, this class is called `StreamingDocServicesCallbacks`.
2. Create an instance of `CMBStreamingDocServices` using the callbacks implemented in step one. In the code sample, this is the `docServices` object. To specify the document engine classes, other than the default, see “Customizing the viewer” on page 537.
3. Create a class that extends `CMBAnnotationServicesCallbacks`. This class is used by `CMBGenericDocViewer` to retrieve and update annotations. You must provide the implementation for the required methods, like `retrieve` and `update`. You can ignore the methods that are not relevant for the type of documents being viewed. In the code sample, this is the `AnnotationServicesCallbacks` class.
4. Create an instance of `CMBAnnotationServices` using the callbacks implemented in step three. In the code sample this is the `annoServices` object.
5. Create an instance of `CMBGenericDocViewer` and initialize it with the `CMBStreamingDocServices`, `CMBAnnotationServices`, and the configuration properties object. Pass null for the properties object if you want to use the default configuration object. The default configuration is constructed from the default configuration file `cmbviewerconfiguration.properties`, contained in `cmbview81.jar`. The message string for this default configuration, like action labels and tool tips, is read from the language specific `cmbViewerMessages.properties` file in `cmbview81.jar`. See “Customizing the viewer” on page 537 for information about configuration options.
6. Add the viewer to the application’s or applet’s main frame.
7. Prepare a menu bar by retrieving the actions from the generic document viewer and add the menu to the application as shown in the section about customizing.
8. Implement the viewer listener interfaces, such as `CMBGenericDocSaveListener` and `CMBGenericDocClosedListener`. Remember to use the `genericDocSave` method of `CMBGenericDocSaveListener` to save a document and annotation set.
9. Add the viewer listeners using the add listener methods of `CMBGenericDocViewer`, such as `addDocSaveListener`.
10. The `CMBStreamingDocServices` object relies on document engine classes to handle the manipulation of images. You can specify the engines and their order using the engine property file, called `cmbviewerengine.properties`, which is contained in `cmbview81.jar`. You can also do this programmatically as demonstrated in the following code snippet:

Specifying the engines

```
// Create engine properties
engineProperties = new Properties();
engineProperties.put("ENGINES", "2");
engineProperties.put("ENGINE1_CLASSNAME",
    "com.ibm.mm.viewer.mstech.CMBMSTechDocumentEngine");
engineProperties.put("ENGINE2_CLASSNAME",
    "com.ibm.mm.viewer.CMBJavaDocumentEngine");
// Create a streaming doc services object and associate
// the document engine classes
docServices = new CMBStreamingDocServices(
    new StreamingDocServicesCallbacks(), engineProperties);
```

Working with documents and annotations

This section provides instructions for using the viewer toolkit components to work with documents and annotations. Complete the following steps to load the document and any annotations, and to view a document in the viewer:

1. Call `loadDocument` in `CMBStreamingDocServices` to load a document into document services and return an instance of `CMBDocument`. Document services is the non-visual layer of the viewer and provides a model for working with the document, visually or non-visually.
2. Call `loadAnnotationSet` in `CMBAnnotationServices` to load any annotations into annotation services and return an instance of `CMBAnnotationSet`. Annotation services provides a model onto the annotation set for a document and allows the annotations to be manipulated both visually and non-visually.
3. Call `showDocument` in `CMBGenericDocViewer` to show the document in the generic doc viewer. The generic doc viewer is the visual component for the viewer.
4. Call `terminate` in `CMBGenericDocViewer` to close the viewer application. This closes all the documents and cleans up the user interface.

Documents and annotations sample

```
// Load the document into document services
CMBDocument document =
    docServices.loadDocument(
        inStream,
        nParts,
        mimetype,
        mimetype,
        null,
        null);

int position = document.getAnnotationPosition();

// Load the annotations into annotation services.
CMBAnnotationSet annotationSet =
    annoServices.loadAnnotationSet(
        annoStream,
        "application/vnd.ibm.modcap",
        position,
        1,
        0);

// Show the document in the generic doc viewer.
genericDocViewer.showDocument(document, annotationSet,
    filename);
```

You can customize the default configuration object, `CMBViewerConfiguration.properties`, located in the `cmbview81.jar` file, or you can create a new configuration object. Whether you create a configuration object or customize `CMBViewerConfiguration.properties`, you must keep the same file name and place it before `cmbview81.jar` in the class path.

Customizing the viewer

A viewer, provided by `CMBGenericDocViewer`, is included with the product. You can customize the functionality provided by the viewer through the configuration properties file. This file contains a series of parameters that control what toolbars display, where they display, and the tool buttons that are available. You can also customize such features as popup menus and the default display resolution.

You can customize the viewer by providing a custom `configProperties` object while constructing the viewer. To view an example of a customized viewer, see the `TGenericDocViewer` viewer application sample. The sample viewer source code, `TGenericdocviewer.java`, and the custom property file it uses, `TViewerDefaultConfiguration.properties` are located in the samples directory. When reading through this section, reference the `TViewerDefaultConfiguration.properties` often.

The viewer configuration contains default toolbars, like **Operation**, **Annotation**, and **Page**, a number of default tools, and a few properties that control certain viewer behaviors like page manipulation, multiple selection, and so forth. By using a custom viewer configuration file, you can control which toolbars are visible, customize existing toolbars, create new toolbars and actions, and enable or disable configurable viewer behaviors. The following code snippet demonstrates how to provide a custom `configProperties` object to the viewer.

Providing configProperties object to the viewer

```
//Load the custom configuration file.
Properties customProperties = null;
try {
    URL url =
        this.getClass().getClassLoader().getResource(
            "TVIEWERDefaultConfiguration.properties");
    InputStream configFile = null;
    Properties defaultProperties = null;
    if (url != null) {
        configFile = url.openStream();
        defaultProperties = new Properties();
        defaultProperties.load(configFile);
        properties = defaultProperties;
    }
}
catch (Exception e) {
    System.out.println(e);
    e.printStackTrace();
}
// Create the generic doc viewer and add it to the window
genericDocViewer =
    new CMBGenericDocViewer(docServices, annoServices,
        customProperties);
```

The following code snippet is the section of the TVIEWERDefaultConfiguration.properties file that customizes the toolbar. By using the various properties, you can customize which toolbars appear, their position, the tools that appear, the order in which they appear, and their groupings. The toolbarname.position property specifies the position of the toolbar with respect to the document. The thumbnail bar position is also specified in the same way. The toolbarname.tools property specifies the list of tools, separated by a comma, that appear on the toolbar. The tools appear on the toolbar in the same order that you specify them. To group tools together use the key word separator.

Customizing the toolbar

```
//The Toolbar defines all the toolbars listed
Toolbars=OperationToolbar,PageToolbar,AnnoToolbar

//You can customize the position and the tools of each toolbar
OperationToolbar.position=NORTH
OperationToolbar.tools=new_doc,open_doc,save_doc,save_as,separator,
export_doc,print,separator,cut,copy,paste,separator,undo,redo,separator,
rotate_90,rotate_180,rotate_270,rotate_pages,separator,zoom_in,zoom_out,
zoom_custom,separator, \

fit_height,fit_width,fit_window,fit_actualsize,separator,enhance,invert,
separator,hide_show,showhidethumb,separator,close_doc,
close_all_doc,separator,help

AnnoToolbar.position=WEST
AnnoToolbar.tools=selectArea,pointer,separator,Arrow,Circle,Highlight,
Line>Note,Pen,Rect,Stamp,Text,eraser,separator, \
move_front,send_back,properties

PageToolbar.position=SOUTH
PageToolbar.tools=page_first,page_prev,goto_page,page_next,page_last,
separator, \
doc_first,doc_prev,doc_next,doc_last

Thumbnailbar.position=EAST
```

All the tools listed in the *Customizing the toolbar* code snippet, except `open_doc`, are default actions and the viewer implements them. The `open_doc` tool is a custom tool that you can add. You can also specify where the tool appears by adding the `open_doc` tool to the `OperationToolbar` after the `new_doc` tool. The following code snippet demonstrates how to add a custom tool:

Adding a custom tool

```
open_doc.label=Open
open_doc.tooltip=Open Document
open_doc.icon=OpenDocument_normal.gif
open_doc.key=control 0
```

After you add the custom tool, the viewer creates an action and a button for the tool but does not have an implementation for the action. You must provide the code to implement the `open_doc` action. The following code snippet demonstrates how to implement the action:

Implementing an action

```
// Add a toolbar button for opening documents
// Get the button for the 'open_doc' tool
JButton button = (JButton)
    ((CMBViewerToolbar) genericDocViewer.getToolBar(
        "OperationToolbar")).getComponent(
        "open_doc");
//Create a new AbstractAction providing implementation for
// actionPerformed
//This implementation calls openDocument() implemnted elsewhere
Action openDocAction = new AbstractAction("open_doc") {
    public void actionPerformed(ActionEvent event) {
        openDocument(null);
    }
};
//Add newly created AbstractAction as ActionListener on the button
button.addActionListener(openDocAction);
// Register Keystrokes for this action
genericDocViewer.registerKeyboardAction(
    openDocAction,
    KeyStroke.getKeyStroke(KeyEvent.VK_O, KeyEvent.CTRL_MASK,
false),
    JComponent.WHEN_IN_FOCUSED_WINDOW);
genericDocViewer.registerKeyboardAction(
    openDocAction,
    KeyStroke.getKeyStroke(KeyEvent.VK_O, KeyEvent.CTRL_MASK,
false),
    JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
button.setEnabled(true);
button.requestFocus();
```

You can add default and custom viewer actions to other controls like menus in your application. The following code snippet demonstrates how to add a viewer action to the menu bar of an application:

Adding viewer actions to an application

```
//create a menubar and menu
JMenuBar mainMenuBar = new JMenuBar();
JMenu viewerMenu = new JMenu("Viewer");
mainMenuBar.add(viewerMenu);
// get an action object that has been loaded into the viewer
Action zoomInAction = genericDocViewer.getAction("zoom_in");
// add the viewer action to the menu
viewerMenu.add(zoomInAction);
// add the menu bar
getRootPane().setJMenuBar(mainMenuBar);
```

The viewer also provides popup menus that you can customize. The popup menus include Page, Thumbnail, Annotation, Document Tab, and Select Area. The names of the popup menus are fixed and signify where the menu appears. The tools available for each popup menu are listed in the value of the `popupmenuName.items` property, which you can customize. The following code snippet is the popup menu section from `TViewerDefaultConfiguration.properties` file:

Customizing popup menus

```
//Popup Menus
//annotation popup
annotationpopup.items=cut,copy,paste,delete,separator,move_front,
send_back,separator,properties
//thumbnail popup
thumbpopup.items=cut,copy,paste,delete,separator,deselectAll,
selectAll,separator,rotate
//page popup
pagepopup.items=cut,copy,paste,delete,separator,rotate,zoom,
fit,navigate,separator,paste
//doc tab popup (appears when right-click on document tabs)

doctabpopup.items=close_doc,close_all_doc,separator,save_doc,print

//select area popup

selectareapopup.items=copy
```

You can add submenus to the popup menus by creating `label` and `item` entries for the values in `popupmenuname.item`. The following code snippet shows how to add submenus for rotate, zoom, fit, and navigate:

Adding submenus

```
//submenu (appear on some of the popup menus)
rotate.label=Rotate Pages
zoom.label=Zoom
fit.label=Fit
navigate.label=Navigate
rotate.items=rotate_90,rotate_180,rotate_270
zoom.items=zoom_in,zoom_out,zoom_custom
fit.items=fit_height,fit_width,fit_window,fit_actualseize
navigate.items=page_first,page_prev,goto_page,page_next,
page_last,showhidethumb
```

Following is a list of other viewer options that you can customize. The code snippet shows how to customize other viewer behavior options. Each of the options are explained in the comments that accompany each property.

- **Inverting documents:** Set the `Document.invert` property to true to automatically invert a document the first time it is opened. Set it to false, so that documents are not automatically inverted when first opened. Note that, by default documents are not inverted.
Example:
`Document.invert=false`
- **Enhancing documents:** To automatically enhance documents upon opening, set the `Document.enhance` property to true. The default is to enhance documents. Note that enhancing certain image documents can increase memory usage.
Example:
`Document.enhance=true`
- **Rotating documents:** The default behavior is to not rotate the document when it is opened. To automatically rotate a document when it is opened, set the `Document.rotate` property according to the following degrees:
 - Do not rotate: `rotate_0`

- Rotate 90 degrees: rotate_90
- Rotate 180: rotate_180
- Rotate 270: rotate_270

Example:

```
Document.rotate=rotate_0
```

- **Showing annotations:** To automatically show all annotations associated with the document when it is opened, set the Annotations.show property to true. The default setting is true. Set the property to false if you want the annotations hidden when the document is opened.

Example:

```
Annotations.show=true
```

- **Zooming:** To zoom in and out on a document, set the Zoom.factor property to a numeric value greater than zero and less than one hundred. This value represents a percentage. The default zoom value is ten percent.

Example:

```
Zoom.factor=10
```

- **Fit-to-window:** To fit the document page to a window, set the Page.fit property according to following options:

- fit_width : Fit pages to view width
- fit_height: Fit pages to the height of the view
- fit_in_window: Fit the entire page within the view
- fit_none: Display the page using the initial zoom (the default)

Example:

```
Page.fit=fit_width
```

- **Resizing Thumbnails:** The ThumbnailSize properties define the width and height of each thumbnail view. The values in the following example are appropriate for an 8.5x11 inch page.

Example:

```
ThumbnailSize.width=60  
ThumbnailSize.height=77
```

- **Manipulating pages:** You can copy, cut, paste, delete, and rotate a page of a document and have all of these changes automatically saved. To save all changes made to a page, set the PageManipulation property to true.

Example:

```
PageManipulation=true
```

- **Selecting multiple pages:** To select more than

Customizing viewer behavior:

```
# MultiplePageSelection enables multi-select of pages within the thumbnails.  
MultiplePageSelection=true
```

```
# Set NewDocument.mimetype property to the MIME type to be used when creating  
# new empty document.  
# Examples include, image/gif, image/jpeg, image/tiff, application/pdf, text/plain,  
# text/richtext, application/vnd.lotus_wordpro  
NewDocument.mimetype=image/tiff
```

```
# Set NewDocument.annotationtype property to a string constant for the  
# type of annotations when creating a new empty document.  
# For example, 'application/vnd.ibm.modcap' for IBM DB2 Content Manager annotations  
# or a two letter representation of the server type such as 'DL','OD','V4'  
NewDocument.annotationtype=application/vnd.ibm.modcap
```


Figure 46 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 Information Integrator for Content 8.1 understands only the DB2 Content Manager annotation format. This annotation format is used by Content Manager Version 7, DB2 Content Manager Version 8. 1, and VI/400 backends.

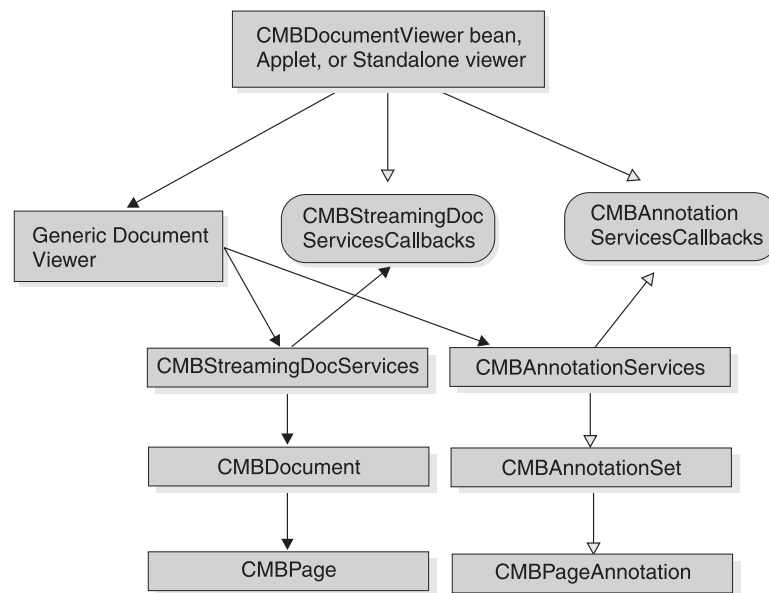


Figure 46. Annotation services class diagram

Understanding annotation editing support

The Model View Controller (MVC) design pattern is used to implement the annotations editing functionality. `CMBAnnotationSet` 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 DB2 Content Manager annotation types that you can create: `CMBArrowAnnotation`, `CMBCircleAnnotation`, `CMBHighlightAnnotation`, `CMBLineAnnotation`, `CMBNoteAnnotation`, `CMBPenAnnotation`, `CMBRectAnnotation`, `CMBStampAnnotation`, and `CMBTextAnnotation`.

Note: You can use annotation services in non-GUI applications.

Building an application using the annotation services

This section includes the steps to follow and APIs to use to build an annotation services application. See the Application Programming 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 samples directory.

Adding a custom annotation type to your application

To add a custom annotation type to the annotation services, complete the steps below. See the Application Programming 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 samples directory. The sample demonstrates how to add a custom annotation type to the annotation services.

Working with the page manipulation functions

The Java viewer toolkit provides capabilities for editing image-paged documents at the page level. This functionality simulates the capabilities of working with paper documents.

Full functionality includes:

- Create, save, and export
 - Create document
 - Save as new document (not exposed in the default toolbar)
 - Save modified document with annotations
 - Export document, with or without annotations
- Multiple selection
 - Multiselect in thumbnails using the mouse or keyboard
 - Select all or deselect all
- Manipulation
 - Cut, copy, and paste selected pages within a document or across documents
 - Delete selected pages
 - Drag and drop selected thumbnail pages to move or copy
 - Permanently rotate pages

Enabling a custom application for page manipulation

You can modify existing custom applications that contain the CMBGenericDocViewer to support page manipulation. By default, CMBGenericDocViewer does not enable page manipulation.

To enable support in a custom application, perform these steps:

1. Set the viewer configuration property for page manipulation and multiple page selection to true. In the CMBViewerConfiguration.properties object, add the following lines:

```
PageManipulation=true
```

```
MultiplePageSelection=true
```

2. Update the viewer configuration to add new toolbar and menu items related to page manipulation. DB2 Content Manager Version 8 Release 3 provides the following actions:

new_doc

Creates a document. The document created within the viewer initially contains no pages. Saving the document creates it on the server.

save_as

Saves the document in the viewer as a new document on the server.

export_doc

Exports the document displayed in the viewer to the file system. This action displays a dialog that allows the user to specify the file name, type, and directory location. A check box enables imprinting the annotations onto the document pages.

3. Create a handler for implementing the CMBGenericDocSaveEvent listener. This handler is registered with the instance of CMBGenericDocViewer. It gains

control whenever a calling viewer action requires saving. The handler also gains control when saving the document when the document is closed.

```
class SaveHandler implements CMBGenericDocSaveListener
{
public void genericDocSave(CMBGenericDocSaveEvent evt)
{
    CMBDocument document = evt.getDocument();
    CMBAnnotationSet annoSet = evt.getAnnotationSet();
    if (evt.getSaveAsNew()) {
        .. logic to save new documents
    }
    if (document.isModified()) {
        OutputStream docstream = ..
        docServices.setPreferredFormats(
            new String[] { document.getMimeType()});
        document.write(docstream);
    }
    if (annoSet.isDirty()) {
        OutputStream annostream = ..
        annoSet.write(annostream);
    }
    ..other save logic
    document.setModified(false);
    annoSet.setDirty(false);
}
}
```

Typically, performing page manipulation operations on documents with annotations modifies the annotations. Analyze the impact of updating a document and its annotations to avoid updating the document and annotations separately if the operations fail.

A CMBGenericDocSaveEvent handler should save both annotations and document content. Prior to V8.3, users used callbacks to save annotations. Callbacks are used when the `isDirty()` flag for a CMBAnnotationSet is true and either there are no registered CMBGenericDocSaveEvent handlers or the registered handlers do not reset the `isDirty()` flag on the annotation set.

If the CMBGenericDocSaveEvent handler saves annotations in all cases, the saving logic in the callbacks is optional.

4. Add logic to enable or disable page manipulation capabilities according to user privileges. The following methods enable or disable page manipulation capabilities in CMBGenericDocViewer:

setPageManipulationEnabled(boolean enable)

Enables or disables all page manipulation capabilities in the viewer.

setPageManipulationPrivilege(CMBDocument document, int privilege, boolean enable)

This method can enable and disable page manipulation on individual documents, and enable and disable the ability to create documents.

5. Add logic to respond to page manipulation operations within the viewer. This is usually necessary if there is additional information that is maintained by the application related to document pages. When users add or remove pages, the application must synchronize the page-specific information with the viewer information.

Deleting pages in a document does not destroy those pages. Deleting removes pages and places them into a removed pages object. Performing a cut operation places a removed pages object on the clipboard. Deleting places this removed pages object on the tasks performed queue, for use in an undo operation. To allow saving application-specific information, CMBGenericDocStateChangedEvent provides `getRelatedInfo` and `setRelatedInfo`

methods, which allows an application to associate a serializable object with pages deleted or copied (using `setRelatedInfo`) and retrieved when the pages are added (using `getRelatedInfo`).

Adding, modifying, or deleting pages triggers example:

`CMBGenericDocStateChangedEvent`.

```
gdv.addDocStateChangedListener(new ChangeHandler());
```

```
class ChangeHandler implements CMBGenericDocStateChangedListener {
public void genericDocStateChanged(CMBGenericDocStateChangedEvent evt) {
if (evt.getChangeType() == CHANGETYPE_PAGES_DELETED) {
.. add logic for pages deleted
} else if (evt.getChangeType() == CHANGETYPE_PAGES_COPIED) {
.. add logic for pages copied (to clipboard)
} else if (evt.getChangeType() == CHANGETYPE_PAGES_ADDED) {
.. add logic for pages added
}
}
}
```

Chapter 14. Working with the JSP tag library and controller servlet

Information Integrator for Content includes a Java Server Pages (JSP) 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 Information Integrator for Content JavaBeans.

This tag library works with the servlet which can act as a controller of a model-view-controller design Web application and performs bean initialization and other actions.

Setting up the tag library and servlet

You must install the tag library and servlet on a Web server with IBM WebSphere Application Server and configure the Web server to use them. For information about installing the tag library and servlet, configuring them, and for information about building WAR/EAR files, see *Planning and Installing Information Integrator for Content*.

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>
      <tr>
        <td><b>Available Search Templates</b></td>
      </tr>
      <tr>
        <td><%= searchtemplate.getName() %></td>
      </tr>
      <tr>
        <td><%= searchtemplate.getName() %></td>
      </tr>
    </table>
  </body>
</html>
```

The taglib directive declares that the page uses the Information Integrator for Content 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 46 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 46. Tag library conventions

Scope	Name	Type	Description
application	connectionPool	CMBConnectionPool	The connection pool bean that is shared across sessions
session	connection	CMBConnection	The instance of CMBConnection for the session
session	schema	CMBSchemaManagement	Schema management bean
session	data	CMBDataManagement	Data management bean
session	user	CMBUserManagement	User management bean
session	query	CMBQuesryService	Query service bean
session	traceLog	CMBTraceLog	Trace log bean; all of the other beans send their trace to this bean
session	docservices	CMBDocumentServices	Document services bean
request	item	CMBItem	The last item that you operated on
request	items	CMBItem[]	Collection of the items that you last operated on
request	searchTemplate	CMBSearchTemplate	The selected search template
request	searchResults	CMBSearchResults	The results from the last search

Tag summary

The following sections summarize the tag library:

Connection related tags

<cmb:datasources" connection="connection">datasource ... </cmb:datasources>

This tag iterates through the available data sources.

connection

Specify the name of a variable of type CMBConnection that contains the connection.

datasource

A string variable to contain the data source name as a string.

Schema related tags

**<cmb:searchtemplates searchTemplates="searchTemplate"> ...
</cmb:searchtemplates>**

This tag iterates through the available search templates.

searchTemplates

Specify the name of an array of type CMBSearchTemplate[] to contain the search templates.

searchTemplate

A variable of type CMBSearchTemplate to contain the search template.

**<cmb:searchcriteria searchTemplate="searchTemplate">criteria ...
</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">criteria ...
</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 CMBDDataManagement.

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 CMBDDataManagement.

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 CMBDDataManagement.

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.

Information Integrator for Content controller servlet

Information Integrator for Content provides a servlet with pluggable actions that can be used when building Web applications. This servlet acts as a controller of a model-view-controller design Web application, performing actions and initializing the beans (the model) which are then accessed in the JSP (the views) either directly or indirectly by using the JSP tags.

Actions are provided for typical application tasks:

- Log on and log off.
- Search.
- Create, retrieve, modify, and delete documents.
- Create folders, and add documents to or remove documents from folders.
- Launch documents and document pages for viewing.

In addition, the servlet performs common tasks before and after the action, such as management of the connection to the content server. After every action, a JSP is invoked to format the results and send them back to the browser.

You can customize the servlet to add new actions and associate JSPs with the actions.

What the servlet can do

Here are some of the aspects of the servlet that you can use:

Connection pooling

The controller servlet uses Information Integrator for Content connection pooling to provide high performance connection management. The time in which a connection is allocated to a session may be either for the request or for the time the session is logged on. Currently connection pooling is at the application scope.

Logon of Timed-Out Sessions

If a session has timed-out, and a request comes into the servlet, the logon JSP is displayed, allowing the user to logon again. The original request is performed after successful logon.

Clean up on session termination

The servlet cleans up the session properly when a session is terminated, either by logging off or by a time-out. This means that the connection is destroyed or returned to the pool. All other Information Integrator for Content 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 `cmbServlet.jsp.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 JSP.

Extending the servlet

All actions known to the servlet are defined in a properties file named `cmbervlet.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 `cmbervlet.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 Information Integrator for Content tag library.

Application parameters

The servlet understands the following application parameters (an alternative is to place these in the `cmbervlet.properties` file).

Application parameter	Values	Description
<code>servletPropertiesURL</code>	URL	The location of the <code>cmbervlet.properties</code> file
<code>defaultServerType</code>	Fed, ICM, OD, DL, V4, IP, DD, ...	Default logon information. This, along with <code>defaultServer</code> , <code>defaultUserId</code> , and <code>defaultPassword</code> 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.
<code>defaultServer</code>		Default logon information.
<code>defaultUserId</code>		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 Information Integrator for Content 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: <ul style="list-style-type: none"> • 0 - log nothing • 1 - log exceptions (default) • 2 -log exceptions, alert messages, WebSphere Application Server headers and attributes, Information Integrator for Content ini files, JVM system properties, Information Integrator for Content internal trace information
connectiontype	0, 1, or 2	The location of the Information Integrator for Content database and content server runtimes: <ul style="list-style-type: none"> • 0 - local (default) • 1 - remote • 2 -dynamic
cmbclient	URL	Location of cmbclient.ini
cmbscs	URL	Location of cmbscs.ini
serviceconnectiontype	0, 1, or 2	Location of services runtimes <ul style="list-style-type: none"> • 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

Application parameter	Values	Description
cachedir	name of a directory	Directory to cache documents during document conversion
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 <i>or</i> 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 `cmb servlet.properties` file. If the action parameter is not specified, and reply is specified, the reply page is executed by the controller servlet after performing standard setup, such as checking the connection for timeout and logon.

Connection Related

action=logon serverType=<> server=<> userid=<> password=<> [connstring=<>] [configstring=<>]

Login to a server. You must specify the server type, server name, user ID, and password. The connect string and init string are optional and are different depending on the type of server.

action=logoff [endSession=<true|false>]

Logout of the server. The session is also ended by default.

Search related

action=searchTemplate template=<> {<criteriaName>.op=<> <criteriaName>=<>}

Perform a search using the specified search template and criteria values.

action=searchEntity entity=<> {attribute.<attrName>.op=<> attribute.<attrName>=<>} [conjunction=<and|or>]

Perform a search using an entity. The attributes values and operators can also be specified. Multiple values in the attribute value are separated with the value delimiter as specified in the application parameters. The attributes are combined together to form a query using `and` (default) or `or`, as specified by the conjunction parameter.

action=searchQuery queryString=<> {queryParameter.<parameterName>=<>}

Perform a search using the specified query string. The query syntax depends on the server being searched.

Any number of additional query parameters may be specified. These are also server dependent.

Item related

action=lock itemId=<>

Lock an item, typically for exclusive access while updating.

action=unlock itemId=<>

Unlock a locked item.

action=createItem type=<document|folder> entity=<>
{attribute.<attrname>=<attrvalue>}

Create an item. If posted, content may be provided.

action=retrieveItem itemId=<>

Retrieve an item's attributes and content. This is useful to insure the latest content as stored on the server is used.

action=updateItem itemId=<> [entity=<>]
{attribute.<attrname>=<attrvalue>}

Update the attributes of an item. If entity is specified, the item is reindexed. Content is also updated if the servlet is invoked through a post.

action=deleteItem itemId=<>

Delete an item.

action=addContent itemId=<>

Add a content part to an item. The content data is posted.

action=getContent itemId=<> contentIndex=<>

Gets the content part and returns it to the browser.

action=updateContent itemId=<> contentIndex=<>

Update a content part on an item; the content data is posted. If no content exists, a content part is added.

action=deleteContent itemId=<> contentIndex=<>

Delete a content part for the specified item.

action=addNoteLog itemId=<>

Modifies the notelog on an item. The text of the notelog is posted.

action=updateNoteLog itemId=<> notelogIndex=<>

Modifies the note log of an item; the text of the note log is posted. If a note log does not exist, it is added.

action=deleteNoteLog itemId=<> notelogIndex=<>

Deletes the note log text of an item. The text of the note log is posted.

Folder related

action=addItemToFolder itemId=<> folderId=<>

Adds the specified item to the specified folder.

action=removeItemFromFolder itemId=<> folderId=<>

Removes the specified item from the specified folder.

Document related

action=viewDocument itemId=<>

Retrieves the document and views it. If the document is paginated, this action forwards to a JSP which generates the viewer frameset. If the document is not paginated, this action returns the actual content of the document.

action=viewPage itemId=<> page=<> scale=<> rotation=<>
annotations=<yes|no>

Retrieves a page of the document.

Servlet toolkit function matrix

Table 47. Servlet function matrix

Action	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
logon	Y	Y	Y	Y	Y	Y	Y
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
removeItem FromFolder	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 48. Servlet function matrix Continued

Action	DB2	JDBC	Fed
logon	Y	Y	Y
logoff	Y	Y	Y
searchTemplate	N/A	N/A	Y
searchEntity	Y	Y	Y
searchQuery	Y	Y	Y
lock	N/A	N/A	Y
unlock	N/A	N/A	Y
createItem	Y	Y	N/A
retrieveItem	Y	Y	Y

Table 48. Servlet function matrix Continued (continued)

Action	DB2	JDBC	Fed
updateItem	Y	Y	Y
deleteItem	Y	Y	Y
addContent	N/A	N/A	Y
getContent	Y	Y	Y
updateContent	N/A	N/A	Y
deleteContent	N/A	N/A	Y
addNoteLog	N/A	N/A	Y
updateNoteLog	N/A	N/A	Y
deleteNoteLog	N/A	N/A	Y
addItemToFolder	N/A	N/A	N/A
removeItem FromFolder	N/A	N/A	N/A
viewDocument	Y	Y	Y
viewPage	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 15. Troubleshooting

This section contains the following troubleshooting topics:

- Compiling a C++ application that is Unicode enabled
- Creating reference attributes
- Updating a resource item
- Importing objects from XML

Receiving an error when compiling C++ applications that are Unicode enabled

Possible cause

The Content Manager V8 C++ APIs do not support Unicode

Action

When compiling C++ applications that use the Content Manager (CM) Version 8 C++ APIs, you must compile with the Unicode flag set to OFF.

To store objects in a Content Manager Unicode enabled database using the C++ APIs, you must meet the following conditions:

- The workstation where the client is running must have at least fix pack 8 of the DB2 Version 7 Run-Time Client installed.
- You must handle string conversion in your application. For example, right to left translation and double byte handling. All the CM C++ APIs can handle ASCII byte strings using a zero terminated character string.
- The workstation must be running the local language. The DB2 Run-Time Client takes the zero terminated character string in the local language from the CM C++ API and stores it in the Unicode enabled library server and resource manager.
- When working with XDOs, some APIs have a code page parameter that must be set to the CCSID (code page number).

Displaying an item's attribute descriptions in the local language on a client application:

The CM product installs using English as the default language. However, after the installation is complete, you can use the system administration client to define other languages that already exist in the system. When you create a new attribute or item type, you have the option to display the attribute or item type name in any of the languages that are defined in the system.

Receiving an error when using reference attributes

Symptom

You are using a reference attribute and receive the following error when you update or add an item: "Unexpected SQL Error (RC 7015)" with (Ext) SQL RC of '-910'

Possible causes

In general, you cannot use reference attributes to connect with an item of the same item type. References are meant to reference items of different item types. You can create a reference to an item of the same item type by setting and modifying the reference attribute on an item that is persistent. You must make persistent the modifications to the reference attribute value only.

An Unexpected SQL Error with SQL RC of -910 occurs if changes relating to the reference attribute are not made independently of other changes.

Action

To create a reference attribute, complete the following steps:

1. Create DDOs A and B.
2. Set any other values in A and B, including adding child components.
3. Add both A and B to the datastore.
4. Check out or lock A.
5. Set A's reference attribute to B.
6. Update A.

Cannot add, store, retrieve, or update a resource item

Symptom

You encounter errors when you attempt to add, store, retrieve, or update resource items (items that are stored in the resource manager) or DB2 Content Manager document model items with parts.

Possible cause

The most common problems with being unable to store resource items is that the resource manager you are attempting access is not running or is unavailable.

Action

Verify that the resource manager you are trying to access is running. To do so, modify the sample URL below to point to your resource manager by replacing smith.stl.ibm.com with the host name of your resource manager. If your resource manager has a different port number (check with your system administrator), replace the number 80 in the URL with that number to point to your resource manager.

Sample URL: `http://smith.stl.ibm.com:80/icrm/ICMResourceManager`

Paste the URL into a Web browser. If a Web page appears, the resource manager server is running. You should see a message similar to the one shown below:

IBM Content Manager V8

Your request: null

Return Code: 9716

Cannot import a DKDDO object from XML

Symptom

You are receiving an `IllegalArgumentException` with the message DGL0303A: Invalid parameter

Possible cause

You are using the wrong date, time, or timestamp format.

Action

See the Online Programming Reference and ensure that the date, time, and timestamp values in your XML files follow the `DKDate`, `DKTime`, and `DKTimestamp` formats documented in their `valueOf()` method.

If you receive the error in the example below, your time value is not valid. Ensure that the time value is specified in the format: hh.mm.ss. Ensure that you used periods (.) instead of colons (:).

Example Error:

```
java.lang.IllegalArgumentException: DGL0303A: Invalid parameter at
com.ibm.mm.sdk.common.DKTime.valueOf(DKTime.java:98) at
com.ibm.mm.sdk.common.DKXMLUtil.processElemDataValue(DKXMLUtil.java:2045)
... at com.ibm.mm.sdk.common.DKXMLUtil.xmlImport(DKXMLUtil.java:237) at
com.ibm.mm.sdk.common.DKDDO.fromXML(DKDDO.java:285) at
XMLImporter.main(XMLImporter.java:32)
```

Receiving an error when updating, reorganizing, or using text indexes for text searchable components

Symptom

You receive the following error when updating, reorganizing, or using text indexes for text searchable components: `DKUsageError: DGL5203A: The password is invalid for the user ID used to administer text indexes.; ICM7172: The password provided is invalid for this user ID, or it is NULL. (STATE) : [LS RC = 7172, SQL RC = -1]`.

Possible cause

The DB2 Text Information Extender (DB2 UDB Version 7) or DB2 Net Search Extender (DB2 UDB Version 8) password is either not set, or set incorrectly. Reset the password in the DB2 Content Manager system administration client.

Action

Complete the following steps to set the password:

1. Log on to the system administration client.
2. Expand the Library Server Parameters category in the left pane.
3. Select Configurations in the left pane. The library server configuration properties display in the right pane.
4. Click the Features tab.
5. Enter the correct DB2 Text Information Extender or DB2 Net Search Extender user ID and password.

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 grant 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 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM	Domino	OS/390
400	Domino.Doc	OS/400
Advanced Function Presentation	HotMedia	PAL
AFP	ImagePlus	QBIC
AIX	IMS	RACF
AIX 5L	iSeries	Redbooks
AS/400	Language Environment	RS/6000
CICS	Lotus	SecureWay
Cloudscape	Micro Channel	SmartSuite
DataJoiner	Mixed Object Document Content Architecture	Tivoli
DB2	MO:DCA	VideoCharger
DB2 Connect	MQSeries	VisualAge
DB2 Extenders	MVS	VisualInfo
DB2 Universal Database	MVS/ESA	WebSphere
developerWorks	NetView	WordPro
DFSMSdfp	Notes	z/OS
DFSORT	OS/2	

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds 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 DB2 Content Manager system. You use access control lists to control user access to *search templates* in the DB2 Information Integrator for Content system.

action. (1) In DB2 Content Manager document routing, specifies how a user can manipulate the *work packages* at a *work node*. DB2 Content Manager provides some actions and you can create your own. Actions must be included in an *action list* before you can apply them to a work node. (2) In DB2 Information Integrator for Content, specifies how a user can manipulate the *work items* at a *node* in the *workflow*. DB2 Information Integrator for Content provides some actions and you can create your own. Actions must be included in an *action list* before you can apply them to a *node*.

action list. (1) In DB2 Content Manager document routing, a set of *actions* that a user can perform on work packages at a work node. The actions that you specify in the action list display as menu choices for the *work packages* in the client users' *worklists*. (2) In DB2 Information Integrator for Content, a set of *actions* that a user can perform on work items in a *workflow*. The actions that you specify in the action list display as menu choices for the *work items* in the client users' *worklists*.

ad hoc process. In DB2 Content Manager document routing, a one step *process* that you define, usually to "link" two other processes.

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 DB2 Information Integrator for Content, 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 DB2 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. See also *character large object (CLOB)*.

BLOB. See *binary large object*.

business application. In DB2 Content Manager document routing, a *work node* that directs work to an external business application that you develop. The business application work node has an identified DLL

or shared library that runs on the server and can launch an external business application, such as a CICS or IMS™ program.

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.

character large object (CLOB). A sequence of characters (single-byte, multibyte, or both) with a size ranging from 0 bytes to 2 gigabytes less 1 byte. See also *binary large object (BLOB)*.

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 DB2 Content Manager APIs to customize a user interface. An application written with the object-oriented or Internet APIs to access *content servers* from DB2 Information Integrator for Content.

Client Application for Windows. A complete object management system provided with DB2 Content Manager and written with DB2 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.

CLOB. See *character large object*.

collection. A group of objects with a similar set of management rules.

collection event list. In DB2 Information Integrator for Content advanced workflow, the criteria that a collection point uses to determine which route the federated folder must follow. Corresponds to a specified amount of time or a list of events, which are *event nodes*. Each collection point must have at least two collection event lists: one with a timeout period (the timeout route) and the other with a list of events that must occur for a federated folder to proceed (event-driven route).

collection point. (1) In DB2 Content Manager document routing, a special *work node* at which a folder waits for arrival of other document or folders, but which does not correspond to a business task. (2) In DB2 Information Integrator for Content advanced workflow, a special *node* at which a *federated folder* waits for arrival of other objects, such as documents or folders, or for specified conditions to be met. A collection point consists of one collection node, one to 20 *event nodes*, and two or more connectors called *collection event lists*.

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 also *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 DB2 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 repository for multimedia, business forms, documents, and related data, and the related metadata required for users to work with that content. DB2 Content Manager and 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 DB2 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 DB2 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 DB2 Content Manager systems and users as a separate unit. An item with the document *semantic type* is expected to contain information that forms a document, but does not necessarily imply that it is an implementation of the DB2 Content Manager document model.

An item created from a document classified item type (a specific implementation of the DB2 Content Manager document model), must contain document parts. You can use document classified item types to create items with either the document or folder semantic type.

Document parts can include varied types of content, including for example, text, images, and spreadsheets.

document content architecture (DCA). An architecture that guarantees information integrity for a document being interchanged in an office system network. DCA provides the rule for specifying form and meaning of a document. It defines revisable form text (changeable) and final form text (unchangeable).

document routing process. In DB2 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.

event node. In DB2 Information Integrator for Content advanced workflow, the set of criteria that specifies the objects or conditions that are required by a collection node. Each collection point can include up to 20 event nodes.

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. A DB2 Information Integrator for Content 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 DB2 Content Manager and to an attribute, policy ID, in 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 DB2 Content Manager.

federated entity. A DB2 Information Integrator for Content metadata object that is comprised of *federated attributes* and optionally associated with one or more *federated text indexes*.

federated folder. In DB2 Information Integrator for Content, a special-purpose folder that stores native entities from one or more content servers.

federated search. A query issued from DB2 Information Integrator for Content that simultaneously searches for data in one or more *content servers*, which can be heterogeneous.

federated text index. A DB2 Information Integrator for Content metadata object that is mapped to one or more *native text indexes* in one or more *content servers*.

file system. In UNIX systems, the method of partitioning a hard drive for storage.

folder. An *item* of any *item type*, regardless of classification, with the folder *semantic type*. Any item with the folder semantic type contains specific folder functionality that is provided by DB2 Content Manager, in addition to all non-resource item capabilities and any additional functionality available from an item type classification, such as *document* or resource item. Folders can contain any number of items of any type, including documents and subfolders. A folder is indexed by *attributes*.

folder manager. The DB2 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 DB2 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 DB2 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 DB2 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 DB2 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 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 DB2 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 DB2 Information Integrator for Content 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.

DB2 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 DB2 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 DB2 Content Manager system that stores, manages, and handles queries on *items*.

link. A directional relationship between two *items*: the source and the target. You can use a set of links to model one-to-many associations. Contrast with *reference*.

local area network (LAN). A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

M

machine-generated data structure (MGDS). (1) An IBM structured data format protocol for passing character data among the various 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 object class. Classification that describes the data that is contained in an object and how you can act on that data. DB2 Content Manager provides four predefined media object classes: DKLobICM, DKStreamICM, DKTextICM, and DKVideoStreamICM. Similar to *MIME type*.

media server. An AIX-based component of the DB2 Content Manager system that is used for storing and accessing video files.

member function. C++ operators and function that are declared as members of a class. A member function has access to the private and protected data members and member functions of an object of its class.

method. (1) In Java design or programming, the software that implements the behavior specified by an operation. (2) Synonym for *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 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 DB2 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) . An Internet standard for identifying the type of object being transferred across the Internet. See also *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 DB2 Content Manager content server, whereas the field policy ID might be a native attribute in a DB2 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, DB2 Content Manager *index classes* are native entities comprised of DB2 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 DB2 Content Manager content server.

network-attached storage. A technology in which an integrated storage system is attached to a messaging network that uses common communications protocols, such as *TCP/IP*.

network table file. A text file that contains the system-specific configuration information for each node in a DB2 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.

node. In DB2 Information Integrator for Content advanced workflow, a generic term for any discrete point in a workflow process.

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 DB2 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 database *object* in a specific way. Privileges include 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*.

process. In DB2 Content Manager document routing, a series of steps through which work is routed. A process contains at least one start node, one *work node*, and one stop node.

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 DB2 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 DB2 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 DB2 Content Manager, *attribute* values that are used to retrieve a stored *item*. In DB2 Information Integrator for Content, 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 DB2 Content Manager; users can also define their own semantic types.

server definition. The characteristics of a specific *content server* that uniquely identify it to DB2 Information Integrator for Content.

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 DB2 Information Integrator for Content.

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 DB2 Content Manager system that installs all of the components of a DB2 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:

- Fixed disk
- VideoCharger
- Media archive
- Tivoli Storage Manager (including optical, stream, and tape)

storage group. Associates a *storage system* to a *storage class*.

storage system. A generic term for storage in the DB2 Content Manager system. See also *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.

subprocess. In DB2 Content Manager document routing, an existing process that you define to run within another process.

sub-workflow. In DB2 Information Integrator for Content advanced workflow, an existing workflow process that is checked in to the workflow server that you define to run within another workflow.

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 DB2 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). In earlier Content Manager versions, the list of *documents* and *folders* that are contained in a folder or *workbasket*. Search results are displayed as a folder table of contents.

TCP. See *Transmission Control Protocol*.

TCP/IP. See *Transmission Control Protocol/Internet Protocol*.

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*.

Transmission Control Protocol (TCP). A communications *protocol* used in the *Internet* and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It uses the *Internet Protocol (IP)* as the underlying protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP). The suite of transport and application *protocols* that run over the Internet Protocol.

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 DB2 Content Manager. This term generally refers to users of client applications, rather than the developers of applications, who use the DB2 Content Manager APIs. In DB2 Information Integrator for Content, anyone who is identified in the DB2 Information Integrator for Content 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 DB2 Information Integrator for Content user IDs and passwords to corresponding user IDs and passwords in one or more content servers. User mapping enables single logon to DB2 Information Integrator for Content and multiple *content servers*.

utility server. A DB2 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

virtual node. In DB2 Content Manager document routing, a distinguishable point within your *process* diagram at which no work is performed or decisions made, but which is required to effectively render the process flow. Start, stop, split, and join are virtual nodes.

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.

work basket. (1) In DB2 Content Manager document routing, a location at which work waits for action by a user or an application. The action can either be taken on the work waiting at the work basket, or the action can be routing the work to another *work node*. (2) In Content Manager Version 7 workflow, synonymous with *worklist*.

workflow. (1) In earlier DB2 Content Manager, a sequence of *workbaskets* through which a *document* or *folder* travels while it is being processed. (2) In DB2 Information Integrator for Content, a series of *work steps*, and the rules governing those steps, through which work is routed. A workflow process contains at least one start node, one *work node*, and one stop node.

workflow coordinator. In earlier DB2 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 DB2 Content Manager workflow and DB2 Information Integrator for Content advanced workflow, a document or object that a user requires to complete a *workflow* activity.

work node. (1) In DB2 Content Manager document routing, a step within a process at which items wait for actions to be completed by end users or applications, or through which items move automatically. Generic term for one of the following three types of work nodes: work basket, collection point, and business application. (2) In DB2 Information Integrator for Content advanced *workflow*, a step within a workflow where work is performed by specified users or groups.

worklist. (1) In DB2 Content Manager document routing, a filter of available *work packages* that are assigned to a user or group. (2) In DB2 Information Integrator for Content advanced workflow, a filter of available *work items* that are assigned to a user or group.

work package. In DB2 Content Manager document routing, a system-defined object that references the item that a user works on during a process. In addition to the item ID, the work package contains additional information that identifies the process to which it belongs and its priority, state, and resume time (if suspended). The user is unaware of a work package because the user works on the referenced item, not on the work package itself.

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 step. (1) A discrete point in a DB2 Content Manager document routing process through which an individual *work package* must pass. (2) A discrete point in a DB2 Information Integrator for Content *workflow* through which an individual *work item* must pass.

X

XDO. See *extended data object*.

XML. See *Extensible Markup Language*.

Index

A

- access control
 - assigning a list to an item 165
 - Content Manager Version 8 113
 - defining lists (ACLs) 161
 - diagram 113
 - lists (ACL) 113, 115
 - NoAccessACL 116
 - PublicReadACL 116
 - retrieving and displaying lists (ACLs) 163
 - rules for lists (ACLs) 115
 - SuperUserACL 116
 - working with 156
 - working with lists for document routing 247
- ActionListName 495
- actions
 - accessing lists 394
 - creating 394
 - DKWorkflowActionICM 227
 - listing in XML 501
 - predefined 502
 - workflow 223
- ad hoc routing 245
- add, BLOB 377
- AddItemToFolderRequest 477
- additional parameters field, OnDemand 327
- addObject 371, 376
- addToFolder 380
- administration
 - object
 - retrieving 175
 - privileges 114
- administration database
 - schema mapping 1
- administration metadata
 - importing and exporting 433
- AIX
 - shared objects for C++ 17
 - starting the RMI server 16
- AllPrivSet 114, 116
- annotation scope
 - GLOBAL 443
 - LOCAL 443
 - VIEW 443
- annotations
 - adding an object to an XDO 43
 - beans 408, 410
 - constant 132
 - customizing 545
 - dkAnnotationExt class 380
 - documents 529
 - editing support 544
 - engine 532
 - OnDemand 338
 - semantic type 132
 - services 543
 - services classes 530
- anyA, DKAny 76

- application building
 - creating in Content Manager Version 8 119
 - non-visual beans 413
 - planning 116
 - using annotation services 545
 - visual beans 425
- application programming interfaces (API)
 - Java 13
 - architecture 13
 - differences from C++ 13
 - multiple search 19
 - packaging 14
- application programming interfaces (APIs)
 - capabilities 13
 - concepts 1
 - online reference 118
 - software components 119
- application programming reference (APR) 118
- ApplicationName 503
- arithmetic query
 - operations 200
 - syntax 218
- ASCENDING 216
- attachments
 - DIME 527
 - MIME 526
- attributes
 - accessing in a DDO 32
 - beans 408, 551, 552
 - comparing with DDOs and XDOs 3
 - Content Manager Version 8 109
 - creating in CM 8 125
 - defining 260
 - definition 109
 - deleting for DKAny 79
 - displaying in beans 414, 422
 - grouping 109
 - image search 307
 - listing for an item type 128
 - listing for OnDemand 328
 - listing in CM for AS/400 347
 - listing in IP OS/390 340
 - listing in relational databases 362
 - managing groups in CM8 126
 - managing in the controller
 - servlet 558
 - modifying for an item 136
 - multi-valued 109
 - querying 200
 - reference 190
 - representing through DDOs 120
 - retrieving for a document item type 175
 - retrieving properties from 34
 - setting and retrieving in CM 8 132
 - specific DK classes for defining 374
 - updating groups in CM 8 127
 - user-defined 189

- attributes (*continued*)
 - making text searchable 192
 - versioning 143
- ATTRONLY, ImagePlus for OS/390 347
- AuditComment 502
- authentication
 - Web services 523
- authentication, XML 465
- AuthenticationData 465
- AUTOCOMMIT, relational databases 361
- average color
 - definition 297
 - query example 301
 - valid values 300

B

- BasicExpression 218
- batching XML requests 509
- BatchRequest 509
- BeanInfo 411
- beans 410
 - ancillary 408
 - annotation 408
 - annotations 410
 - batch support 406
 - BeanInfo classes 411
 - building in WebSphere Studio
 - Application Developer 404
 - characteristics
 - customization 404
 - events 404
 - introspection 403
 - methods 404
 - persistence 404
 - properties 404
 - CMBSchemaManagement 405
 - connection 405, 550
 - connection pool
 - CMBConnectionPool 550
 - data management 405, 550
 - data source
 - datasource bean 550
 - document services 410, 550
 - documents 553
 - event and listener classes 411
 - exception classes 411
 - folders 553
 - helper 407
 - introduction 403
 - invoking 405
 - items 550
 - JAR files 404
 - Java Viewer related
 - definition 403
 - locale, changing 412
 - message schema 461
 - Model View Controller (MVC) 406
 - non-visual
 - building applications 413

- beans (*continued*)
 - non-visual (*continued*)
 - categories 407
 - configurations 406
 - considerations 411
 - definition 403
 - events 413
 - features 406
 - introduction 405
 - properties 413
 - other builders 404
 - query service 550
 - requirements 404
 - schema 405
 - schema management 550
 - search criteria 551
 - search request 405
 - search results 550
 - search template 550, 551
 - searching OnDemand 338
 - session listeners 411
 - singletons 411
 - threading 412
 - trace log 550
 - tracing 413
 - understanding basic concepts 403
 - user management 550
 - using in builders 404
 - versioning 414, 423
 - visual
 - attributes editor 422
 - building applications 425
 - collation strength 423
 - connecting 425
 - connection 423
 - default viewers 422
 - definition 403
 - document viewer 420
 - external viewers 422
 - folder viewer 419
 - general behaviors 423
 - help events 424
 - hiding and showing buttons 423
 - introduction 414
 - key events 424
 - logon panel 414
 - names 414
 - overriding pop-up menus 419
 - pop-up menus 419
 - replacing 424
 - save/restore configuration 424
 - search panel 417
 - search results viewer 418
 - search template list 416
 - search template viewer 417
 - specialized behaviors 424
 - text search areas 417
 - tree pane 418
 - using in windows 426
 - versions viewer 423
 - viewer specifications 421
 - web.xml 405
 - workflow 405, 409
 - worklist 405
 - XML 408
 - XML services 461
- BETWEEN 207, 216

- binary data,
 - See* XDO
- binary large object (BLOB) 377
 - adding 377
 - classes 260
 - concatenating 377
 - copying data 377
 - creating XDOs for 37
 - identifying the file handler 378
 - inserting argument data 378
 - managing content 377
 - methods 377
 - opening asynchronously 378
 - retrieving 377
 - returning the length 377
 - searching 377
 - specific DK classes 377
 - specific DKPidXDO classes 381
 - testing with an XDO function 46
 - updating 377
- binders, Domino.Doc 356
- BLOB,
 - See* binary large object (BLOB)
- buffer, adding an XDO 41
- business application node 223
- Business Process Choreographer 518

C

- C# samples 517
- C++
 - constants 22
 - DKAny 72
 - DKConstant.h 13
 - DLL files 17
 - error message property files 382
 - escape sequences 211
 - libraries 17
 - relational database configuration strings 38
 - setting up the environment 17
 - shared objects for AIX 17
 - XML support 13
- cabinets, Domino.Doc 356
 - listing attributes 359
- caching, controller servlet 557
- callback, executing 20, 188
- catalogs, image search 302
- cc2mime.ini 361
- CC2MIMEFILE, relational databases 361
- CCSID 279
- changePassword 122, 371
- ChangePasswordRequest 466
- character large object (CLOB) 379
 - adding content 379
 - classes 378
 - deleting a portion of 379
 - deleting content 379
 - file handler 379
 - retrieving content 379
- checked out item
 - querying 189
 - querying by person 205
 - querying by timestamp 205
 - querying example 205
- checkedOutUserId 381
- checkin 481
- checkIn, dkDatastoreExt 380
- CheckinItemRequest 488
- checkout 481
- checkOut, dkDatastoreExt 380
- CheckoutItemRequest 488
- child components
 - Content Manager Version 8 109
 - creating in CM 8 130
 - diagram 109
 - referencing 111
 - representation in the data model 189
 - representing as DDO attributes 120
 - versioning 112
- class name terminology 259
- CLOB,
 - See* character large object (CLOB)
- close, result set cursor 376
- cmadmin.xsd 429
- cmb81.jar 404
- CMBAnnotationPropertiesInterface 546
- CMBAttribute
 - changing locale in display names 412
- cmbcc2mime.ini 556
- cmbclient.ini 556
- cmbcm81.jar 368
- cmbcm81.lib 17
- cmbcm817.dll 368
- cmbcm817d.dll 368
- cmbcm81d.lib 17
- CMBConnection 338, 405, 550
- cmbcs.ini 406, 556
- CMBDataManagement 405, 550
- CMBDocumentServices 533, 550
- CMBDocumentViewer 414, 420
 - specifications 421
 - terminating 421
- CMBEntity
 - changing locale in display names 412
- CMBFolderViewer 414, 419
- CMBItem 550
- CMBItemAttributesEditor 422
- CMBLogonPanel 414
- cmbmessages.xsd 461
- CMBObject 552
- CMBPage 554
- CMBPageAnnotation 544
- CMBQueryService 405, 550
- cmbregist81.bat 16
- cmbregist81.sh 16
- CMBSchemaManagement 550
- CMBSearchPanel 417
- CMBSearchResults 550
- CMBSearchResultsView 338
- CMBSearchResultsViewer 414, 418
- CMBSearchTemplate 550
- CMBSearchTemplateList 338, 414, 416
- CMBSearchTemplateViewer 338, 414, 417
- cmbervlet.properties 555
- CMBServletAction 555
- cmbervletjsp.properties 554
- CMBSpecificWebService.cs 519
- CMBSTCriterion 551
- cmbsvclient.ini 556
- cmbsvcs.ini 556
- CMBTraceLog 550
- CMBUserManagement 550

- cmbview81.jar 534
- CMBViewerConfiguration.properties 534
- CMBXMLAttachment 409
- CMBXMLMessage 409
- CMBXMLMessage bean 461
- cmbxmlservice.jar 429
- CMBXMLServices 409
- cmdatamodel.xsd 429
- CMWebServiceClient.cs 519
- COBRA
 - Persistent Data Service (PDS) 1
 - Persistent Object Service 1
- code page conversion
 - setting console subsystem in Windows 19
- collection point
 - defining 231
 - description 221
- CollectionResumeListEntry 496
- collections
 - deleting 79
 - dkCollection class 370
 - federated 80
 - sorting 79
- collections and iterators
 - C++
 - memory management 78
 - Java 76
 - sequential collection 76
 - sequential iterator 77
- color search
 - average 297, 300, 301
 - histogram 297, 300, 301
 - positional 297, 301
 - texture 298, 301
- com.ibm.mm.sdk.client 13
- com.ibm.mm.sdk.common package 368
- com.ibm.mm.sdk.server 13
- combined query
 - definition 20
- Combined query
 - C++
 - Programming tips 317
 - ranking 317
 - Java 194, 314
 - parametric with text 314
 - using a scope 316
- common EIP classes 369
- common privilege, dkDatastoreExt 381
- CompareOperator 218
- Comparison 217
- concatReplace 377
- configString 465
- configuration strings, relational databases 38
- connection bean 550
- connection pooling
 - servlet 554
 - servlet values 555
- connectors
 - controller servlet values 555
 - customizing 368
- connectString 465
- connectToWorkflow 465
- console subsystem, setting in Windows 19
- constants 22
- constants (*continued*)
 - common 382
 - routing 251
- container variables, workflow 223
- ContainerData 501
- ContainerDefinition 495
- containers
 - constant 132
 - Content Manager Version 8 111
 - semantic type 132
- contains-text-basic, text search 192
- contains-text-db2
 - query example 201
- contains-text-oracle
 - query example 201
- contains-text, text search 193
- CONTENT
 - ImagePlus for OS/390 347
- Content Manager for AS/400
 - index classes 347
 - introduction 347
 - listing entities and attributes 347
 - mapping from federated 6
 - running a query 349
- Content Manager Version 8
 - access control 113
 - attributes 109
 - basic concepts 108
 - checking items in and out 145
 - child components 109
 - components illustration 108
 - connecting to 120
 - controlling access
 - access lists 115
 - user groups 115
 - controlling access privileges 113
 - creating a content server 120
 - creating a document 176
 - creating a password 122
 - creating an application 119
 - creating attributes 125
 - creating item types 122
 - defining a resource item type 138
 - deleting a document 180
 - document parts 109
 - documents 109, 111
 - folders 111
 - insurance scenario sample 119
 - introduction 107
 - item types 109
 - items 108
 - library server 107
 - library server and resource manager
 - consistency 181
 - linking items 154
 - links 110
 - listing attributes for an item type 128
 - listing item types 124
 - managing attribute groups 126
 - managing documents 173
 - mapping from federated 6
 - modifying item attributes 136
 - non-resource items 109
 - objects 110
 - planning an application 116
 - privilege set 114
 - query examples 196
- Content Manager Version 8 (*continued*)
 - querying 187
 - references 110, 111
 - resource items 109
 - resource manager 107
 - retrieving a document 180
 - retrieving items 141
 - root components 109
 - routing documents 221
 - samples 118
 - searching for items 140
 - setting and retrieving item
 - attributes 132
- System Managed Storage (SMS)
 - server 107
- transactions 181
- understanding the query
 - language 166
- understanding the search query 187
- updating documents 178
- versioning 112
- working with access control 156
- working with folders 148
- working with the resource manager 167
- Content Manager, earlier versions
 - handling large objects 261
 - image search 295
 - introduction 261
 - mapping from federated 6
 - persistent identifier (PID) 262
 - representing documents 262
 - representing folders 263
 - representing parts 262
 - updating folders 269
 - updating parts 267
 - workflow 318
- content parts
 - attaching in Web services 526
- content servers
 - accessing information on 260
 - adding a document or folder 371
 - adding members to folders 380
 - beans 407
 - changing password 371
 - checking out documents or
 - folders 380
 - class names 259
 - connecting 371
 - Content Manager for AS/400 347
 - Content Manager Version 8 107
 - Content Manager, earlier versions 261
 - controller servlet values 555
 - creating 259
 - creating a DDO in 371
 - customizing 368
 - data definition hierarchy 260
 - defining 259
 - deleting a document or folder 372
 - disconnecting 371
 - Domino.Doc 355
 - dynamic data object (DDO) 2
 - evaluating a parametric query
 - from 87
 - evaluating a text query from 91
 - evaluating an image query from 310

- content servers *(continued)*
 - extended data object (XDO) 2
 - extending FeServerDefBase for custom connectors 382
 - extending objects 371
 - extension classes 380
 - federated 5
 - getting the common privilege 381
 - identifying where result set cursor belongs 376
 - image search 295
 - ImagePlus for OS/390 340
 - listing function names 380
 - managing binary data 377
 - managing CLOB content 379
 - managing data 1
 - moving a document or folder 372
 - OnDemand 326
 - perform transactions 371
 - persistent identifiers (PIDs) 4
 - querying 371
 - referring to 380
 - registering in federated 7
 - registering mapping information 372
 - relational databases 360
 - relationship to DDOs 3
 - removing members from folders 380
 - result set cursor 260
 - retrieving a document or folder 372
 - retrieving the administration object 175
 - retrieving the form overlay object 381
 - returning names 371
 - returning the user ID objects 371
 - running a parametric query from 85
 - running a text query from 90
 - running an image query from 309
 - servers supported by EIP 1
 - specific DK classes for defining 375
 - updating a document or folder 372
 - user management classes 381
 - using with RMI 16
 - XDO classes 381
- contentOption 470
- ContinueProcessRequest 506
- contrast 298
- controlled entities 113
- controller server
 - defaults 555
- controller servlet
 - actions 554, 555
 - clean up 554
 - connection pooling 554
 - connection pooling values 555
 - conventions 555
 - conversion parameters 557
 - extending 555
 - JSP sets 554
 - locale 554
 - names separator parameter 557
 - pages that show error messages 556
 - parameters 555, 557
 - properties file 555, 557
 - reference 555
 - replay 555

- controller servlet *(continued)*
 - request parameters
 - action 558
 - connection related 558
 - document related 559
 - folders 559
 - general 557
 - items 558
 - managing content 559
 - reply 558
 - search 558
 - service runtimes 556
 - session management 554
 - toolkit function matrix 560
 - tracing 556
 - using 554
- convert, schemas 459
- createChildDDO 130
- createDDO 29
- CreateItemRequest 468
- CreateLinkRequest 490
- creating 131
- criterion, search, bean 551
- cs package 14
- cursor, result set
 - adding elements 376
 - checking validity 375
 - closing 376
 - deleting elements
 - deleteObject 376
 - destroying 376
 - getting content server information 376
 - getting the handle 376
 - opening 376
 - pointing to an element 376
 - positioning 375
 - retrieving elements 376
 - scrolling 375
 - updating 375
 - updating elements
 - updateObject 376
- custom connectors
 - developing 368
 - extending the FeServerDefBase class 382
 - system administration 369

D

- data definition classes 260
- data instances
 - importing and exporting 454
- data items, DDO 31
- data model
 - applying the query language 189
 - representing in XML 432
 - representing query in XML 197
- data model objects
 - importing and exporting 434
- data_id 34
- databaseNameStr 121
- datastore,
 - See* content servers
- DB2
 - client configuration assistant 17
 - client support 17

- DB2 *(continued)*
 - configuration strings 38
- DB2 DataJoiner 360
 - configuration strings 38
- DB2 Extenders 2
- DB2 Net Search Engine (NSE) 20
- DB2 Text Information Extender
 - Java
 - Boolean query 275
 - Free text query 275
 - GTR query 276
 - Hybrid query 275
 - Load and index data 284
 - Programming tip 278
 - Proximity query 276
 - Setting heap size 261
- ddoDocument 43
- decision point 222
- decision points 225
- del, BLOB 377
- deleteDKAttributeDef 79
- DeleteItemRequest 487
- DeleteLinkRequest 490
- deleteObject 371, 372
- deleteSchemaMapping 460
- deleting 377
- DemoSimpleAppl.java 413
- DESCENDING 216
- destroy, result set cursor 376
- DfltACLCode 116
- diagnostic information,
 - See* tracing
- diagram definition 225
- DIGIT 217
- Direct Internet Message Encapsulation (DIME) 527
- directionality 298
- display flag, foreign key 134
- display names
 - changing the language of 412
- DisplayName 502
- displayNamesEnabled 412
- DIV 216
- DK_CM_DOCUMENT 262
- DK_CM_FOLDER 263
- DK_CM_OPT_ACCESS_MODE 260
- DK_CM_PROPERTY_ITEM_TYPE 262, 263
- DK_CM_READWRITE 260
- DK_CM_VERSION_LATEST 142
- DK_DL_OPT_ACCESS_MODE 377
- DK_OPT_TS_CCSSID 279
- DK_OPT_TS_LANG 279
- DK_READWRITE 377
- DK_SS_CONFIG 320, 321
- DK_SS_NORMAL 320, 321
- DK_TS_DOCFMT_HTML 285
- dkAbstractWorkflowUserExit 394
- dkAnnotationExt 380
- DKAny
 - assignment from 74
 - assignment to 74
 - checking the type 75
 - deleting collections 79
 - destroying 75
 - display of 75
 - memory management 73

- DKAny (*continued*)
 - programming tips 76
 - type code, getting 74
 - Typecode 73
 - using inside a collection 76
 - using type constructors 73
- dkAttrDef 260
 - classes 374
- DKAttrFieldDefDD 356
- DKAttrGroupDefICM 127
- DKAttrKeywordDefDD 356
- DKAttrProfileDefDD 356
- DKBinderDefDD 356
- dkBlob 260
 - classes 377
 - methods 377
- DKBlobDL
 - setToBeIndexed 311
- DKCabinetDefDD 356
- dkClob
 - classes 378
- dkCollection 370
- DKCollectionResumeListEntryICM 226
- DKConstant 22, 382
- DKConstant.h 13
- dkCQExpr 370
- dkDatastore
 - addObject 371
 - changePassword 371
 - commit 371
 - connect 371
 - deleteObject 371, 372
 - disconnect 371
 - evaluate 371
 - execute 371
 - executeWithCallback 371
 - introduction 371
 - listDataSourceNames 371
 - listDataSources 371
 - listMappingNames 372
 - moveObject 372
 - registerMapping 372
 - registerServices 371
 - retrieveObject 371, 372
 - rollback 371
 - unRegisterMapping 372
 - unregisterServices 371
 - updateObject 372
- DKDatastore 259
 - custom connectors 369
- DKDatastorexx 259
- dkDatastoreDef 259
 - classes 372
- DKDatastoreDef
 - methods 372
- DKDatastoreDefDL
 - additional functions 373
- DKDatastoreDefOD
 - additional functions 373
- DKDatastoreDL
 - Java 23
 - connecting 23
 - DKDatastoreDL options 24
 - list schema and schema attributes 26
 - List servers 25
 - dkDatastoreExt
 - classes 380
 - functions 380
 - DKDatastoreICM 119
 - DKDatastoreIP 340
 - DKDatastoreOD 327
 - listEntities 328
 - DKDatastoreQBIC 296
 - DKDatastoreTS
 - attributes 277
 - Java 274
 - connecting 277
 - DKDatastoreTS options 278
 - list schema 280
 - list servers 279
 - DKDatastoreV4 347
 - DKDatastoreIP
 - listEntityAttrs 343
 - DKDatastoreOD
 - listSearchTemplates 330
 - dkDDO 369
 - DKDDO,
 - See* dynamic data object (DDO)
 - DKDLITEMID 277
 - DKDocRoutingServiceICM 224
 - DKDocRoutingServiceMgmtICM 224
 - DKDocumentConverter 460
 - DKDocumentDefDD 356
 - DKDSIZE 277
 - dkEntityDef 259
 - classes 373
 - functions 373
 - DkEntityDefIP
 - getAttr 341
 - DKEntityDefIP
 - listAttrNames 341
 - DKException 21, 117
 - DKFederatedCollection 80
 - dkFederatedIterator 80
 - dkFederatedQuery 80
 - DKFederatedQuery 8
 - DKFixedView 327
 - DKFixedViewDataOD 327
 - DKFolder 263
 - dkIterator 80
 - DKLink 154
 - DKLinkCollection 156
 - DKLITEMID 307
 - DKMessage_en_US.properties 382
 - DKMessage_en.properties 382
 - DKMessage_es_ES.propertie 382
 - DKMessage_es.properties 382
 - DKMessageId 382
 - DKPARTNO 277, 307
 - DKParts
 - earlier CM 262
 - DKPidXDO 381
 - DKPrivilegeSetICM 158
 - DKProcessICM 224
 - dkQuery 370
 - dkQueryableCollection 370
 - DKRANK 262, 277, 307, 314
 - DKRCNT 277
 - DKREPTYPE 277, 307
 - DKResource 327
 - DKResourceGrpOD 327
 - DKResults 370
 - DKResults (*continued*)
 - positioning the iterator 81
 - dkResultSetCursor 260
 - functions 375
 - DKResumeListEntryICM 227
 - DKRMConfiguration 121
 - DKRoomDefDD 356
 - DKRouteListEntryICM 226
 - DKSchemaConverter 459
 - dkSchemaMapping 372
 - DKSequentialCollection 267, 269, 370
 - dkSequentialIterator 370
 - dkServerDef 260
 - classes 375
 - functions 375
 - dkSort 79
 - DKStorageManageInfo 65
 - DKTimestamp
 - suspending a workflow 389
 - dkUserManagement 381
 - DkViewOD 327
 - DKViews 327
 - DKWorkBasketDL 318
 - DKWorkflowActionICM 227
 - DKWorkflowFed
 - resume 390
 - suspend 389
 - DKWorkflowServiceDL 318
 - DKWorkflowServiceFed
 - svWf 388
 - DKWorkflowServicesFed 385
 - dkWorkflowUserExit 394
 - DKWorkItemFed
 - checkIn 393
 - checkOut 393
 - DKWorkListFed 391
 - DKWorkListICM 226, 235
 - DKWorkNodeContainerDefICM 226
 - DKWorkNodeICM 226, 230
 - DKWorkPackageICM 227, 243
 - dkXDO 378
 - dkXDOBase 369
 - open 378
 - DKXMLeExportList 433
 - DKXMLSysAdminService
 - constants 433
 - methods 432
 - DLL files
 - C++ 17
 - DLLName 503
 - DLSEARCH_DocType 82
 - doAction 394
 - document model,
 - See* document parts
 - Document Object Model (DOM)
 - building from an XML message 521
 - document parts
 - constant 122
 - Content Manager Version 8 109
 - item type 122
 - document routing,
 - See* routing
 - document viewer toolkit
 - annotation services 543
 - applet or servlet 533
 - architecture 530
 - creating a generic viewer 534

- document viewer toolkit (*continued*)
 - customizing the generic document viewer 534
 - dual-mode and applet or servlet 533
 - engines
 - AFP2Web document 531
 - INSO document 531
 - Java document 531
 - MS-Tech Document 531
 - example applications 532
 - introduction 529
 - Java application 532
 - page manipulation 546
 - standalone viewer 532
 - thin client 533
- documents 131
 - adding 371
 - annotating 529
 - beans 410, 550, 553
 - caching in the controller servlet 557
 - checking out and in 380
 - Content Manager Version 8 109, 111
 - creating a document item type 174
 - creating in CM 8 176
 - creating the management data model in CM 8 174
 - deleting 372
 - deleting in CM 8
 - SDocModelItemICM 180
 - differences in managing in earlier CM 268
 - displaying in beans 414, 420
 - Domino.Doc 356
 - earlier CM workflow 318
 - managing in CM 8 173
 - moving 372
 - representing in the data model 190
 - retrieving 372
 - retrieving in CM 8 180
 - routing through a process 221
 - semantic type 132
 - starting a routing process 239
 - text searching for 192
 - updating 372
 - updating in CM 8 178
 - versioning parts in the management data model 180
 - viewing,
 - See* document viewer toolkit
- Domino.Doc
 - introduction 355
 - listing cabinet attributes 359
 - listing entities and subentities 357
 - mapping from federated 6
 - object model 356
 - open document management API (ODMA) 355
 - query syntax 359
 - querying 359
 - querying for DKResults object 82
- DSNAME, relational databases 361, 362
- dsType 338
- dynamic configuration, beans 406
- dynamic data object (DDO)
 - accessing attributes 32
 - accessing folder contents 72
 - adding properties 30

- dynamic data object (DDO) (*continued*)
 - attribute groups in CM 8 126
 - COBRA specification 1
 - comparing with attributes 3
 - dkDDO 369
 - federated 5
 - image search persistent ID 307
 - image search result 302
 - introduction 2
 - obtaining all folders containing 154
 - persistent identifier 4
 - populating with setData 131
 - properties 262
 - referencing 111
 - relationship with content servers 3
 - representing in image search 307
 - representing multimedia content 3
 - retrieveObject 143
 - retrieving attribute properties 34
 - sorting a collection of 79
- Dynamic Data Object (DDO)
 - C++
 - deleting 36
 - Java 28
 - adding 31
 - attribute, DKPARTS 67, 70
 - creating 28
 - data item values 31
 - displaying 35
 - Information, DB2 Text Information Extender 277
 - Information, Digital Library 120, 262
 - PID 30
 - properties 33

E

- EIP workflow services,
 - See* workflow
- Enterprise Information Portal
 - common classes 369
 - concepts 1
 - database infrastructure 369
 - databases 1
 - document engines 531
 - dynamic data objects (DDO) 2
 - extended data objects (XDOs) 2
 - organization diagram 1
 - persistent identifier (PID) 4
 - schema mapping 1
 - supported content servers 1
- Enterprise Java Beans (EJBs) 406
- EnterUserDLL 496
- EnterUserFunction 496
- entities
 - beans 407, 551
 - class hierarchy 259
 - controlled, Content Manager Version 8 113
 - identifying 4
 - ImagePlus for OS/390 346
 - listing in CM 8 124
 - listing in CM for AS/400 347
 - listing in Domino.Doc 357
 - listing in IP OS/390 340
 - listing in relational databases 362

- entities (*continued*)
 - listing in XML 466
 - mapping in federated 5
 - moving items through XML 492
 - moving objects 381
 - native 338, 370
 - OnDemand folders as native 338
 - privileges 114
 - schema mapping 370
 - searching in Domino.Doc 359
 - specific DK class names for defining 373
 - storing federated folders in 11
- ENTITY_TYPE 327, 337
- EntityList 467
- environment, setting
 - C++ 17
 - Java 15
- error messages
 - property files 382
- ErrorCode 117
- ErrorState 117
- escape sequences, query examples 206
- ESCAPE_LITERAL 217
- ESCAPE_KEYWORD, query 217
- evaluating queries 82
- evaluating search 20
- events
 - beans 411
- EXCEPT 206, 216
- exceptions, handling 21
- executeWithCallback 371
- executing queries 82
- executing search 20
- executing search with callback 20, 188
- exponent, query 217
- exporting XML
 - See* extracting XML
- expression class 370
- expression, query 217
- ExpressionList 218
- expressions, query 204
- ExpressionWithOptionalSortBy 217
- extended data object,
 - See* XDO
- Extended Search
 - mapping from federated 6
- Extenders 2
- extensible markup language,
 - See* XML
- extracting XML
 - administration metadata 433
 - data instances 454, 455
 - data model metadata 434
 - exporting object dependencies 457
 - input and output formats 432

F

- FASERVTYPE table 383
- features
 - image search 296
 - maximum results, image search 299
 - name, image search 299
 - value, image search 299
 - weight, image search 299
- federated 11

- federated (*continued*)
 - attributes 5
 - beans 551
 - collection 8
 - collections 80
 - document model 5
 - entities 5
 - folders 11
 - image search 9
 - items 5
 - iterators 80
 - layer for beans 407
 - mapping 5
 - mapping user IDs 7
 - query
 - creating 8
 - examples 9
 - processing illustration 8
 - results 8
 - syntax 9
 - registering content servers 7
 - schema mapping 7
 - searching
 - expression class 370
 - illustration 6
 - introduction 5
 - process 8
 - relational illustration 8
 - server definition base class 382
 - system administration functions 12
- federated document model 5
- FeServerDefBase 382
- fetchNext 376
- fetchObject 376
- fields, Domino.Doc 356
- file, adding an XDO from 42
- findObject 376
- FLoadSampleTSQBICDL 282
- FLOAT_LITERAL 217
- folders
 - accessing 72
 - adding 371
 - adding contents to 149
 - adding members to 380
 - beans 553
 - checking out and in 380
 - Content Manager behavior 72
 - Content Manager Version 8 111
 - creating in CM 8 148
 - deleting 372
 - displaying in beans 414, 419
 - earlier CM workflow 318
 - enabling the mode in
 - OnDemand 337
 - listing in OnDemand 330
 - managing in the controller
 - servlet 559
 - moving 372
 - obtaining all folders with a specific
 - DDO 154
 - OnDemand folders as native
 - entities 338
 - removing contents from 151
 - removing members from 380
 - representing in earlier CM 263
 - retrieving 372
 - retrieving contents 153

- folders (*continued*)
 - semantic type 132
 - updating 372
 - updating in earlier CM 269
 - XML 477
- foreign key attribute values 133
- form overlay object 381
- fromXML() 454
- FTSearch 359
- FunctionName 218, 503

G

- garbage collector, Java 13
- generic document viewer,
 - See* document viewer toolkit
- GenericWebServiceSample.java 521
- getCommonPrivilege 381
- getContent 377
- getContentToClientFile 377, 379
- getDataModelDefs 433
- getDatastore 380
- getDisplayName() 412
- getLocale 412
- getName() 412
- getNonDisplayName() 412
- getOpenHandler 378, 379
- getPosition 375
- GetPrivilegesRequest 476
- getSchemaMappingNames 460
- getStorageSchema 459
- getSysAdminDefs 433
- getXSLTQuery 459, 460
- GLOBAL 443
- graphical user interface (GUI),
 - See* beans, visual

H

- handle, result set cursor 376
- helper beans 407
- Hierarchical Storage Management (HSM) 167
- highlighting, text search 91
 - getting information for a particular
 - result 95
 - getting information for each result 92
- histogram color
 - definition 297
 - query example 301
 - valid values 300
- history
 - constant 132
 - semantic type 132

I

- ICM connector,
 - See* Content Manager Version 8
- ICMBASE
 - representing in Web services 526
- ICMCHECKEDOUT
 - attributes 189
 - example 205
- ICMCHKOUTTS 205
- ICMCHKOUTUSER 205

- ICMLogon 115
- IDENTIFIER 217
- identifiers
 - image search 296, 298
- image search 295
 - applications 298
 - attributes 307
 - average color 297, 300, 301
 - catalogs 296, 297
 - closeCatalog 302
 - connecting to 302
 - creating queries 299
 - databases 296, 297
 - definition 20
 - diagram 296
 - disconnecting from 302
 - evaluating a query from a content
 - server 310
 - features 296, 299
 - federated 9
 - histogram color 297, 300, 301
 - identifiers 298
 - indexing an XDO 311
 - listDatabases 302
 - listing catalogs 304
 - listing databases 304
 - listing features 304
 - listing servers 303
 - loading data to be indexed 311
 - maximum results 299
 - openCatalog 302
 - positional color 297, 301
 - query syntax 299
 - querying 307
 - representing information with a
 - DDO 307
 - running a query from a content
 - server 309
 - search criteria 299
 - texture 298, 301
- ImagePlus for OS/390
 - introduction 340
 - listing attributes 340
 - listing entities 340
 - mapping from federated 6
 - query parameters 346
 - query syntax 345
- imldiag.log 294
- importing XML
 - See* ingesting XML
- inbound links 155
- INBOUNDLINK 190
- indexes
 - text searching on 88
- indexing
 - image search 311
- indexOf 377, 379
- Information Catalog
 - mapping from federated 6
- information center 118
- Information Mining
 - beans support 407
- ingesting XML
 - administration metadata 433
 - constants 433
 - data instances 454, 456
 - data model metadata 434

- ingesting XML (*continued*)
 - input and output formats 432
- initialization parameters field,
 - OnDemand 327
- insert, BLOB 378
- INTERGER_LITERAL 217
- INTERSECT 206, 216
- IS 216
- isBegin 375
- isCheckedOut 381
- isEnd 375
- IsFTIndexed 359
- isInBetween 375
- isOpen 376
- isOpenSynchronous 378, 380
- isScrollable 375
- isSupported 380
- isUpdatable 375
- isValid 375
- item classes,
 - See* item types
- item parts
 - comparing with DDOs and XDOs 3
- item types
 - classifications 122
 - components, root and child 109
 - Content Manager Version 8 109
 - creating a definition 138
 - creating in Content Manger Version 8 122
 - importing and exporting as XML
 - schemas 434
 - listing 124
 - listing attributes for 128
 - querying multiple 200
 - versioning 112, 147
- ItemAdd 115
- ItemAdminPrivSet 114
- ItemQuery 115
- ItemReadPrivSet 116
- items
 - adding to a folder 149
 - beans 408, 550, 552
 - checking in and out 145
 - Content Manager Version 8 108
 - creating a document item type 174
 - creating instances in Web
 - services 524
 - federated 5
 - importing and exporting instances as XML 454
 - linking 154
 - linking through folders 111
 - managing with the controller
 - servlet 558
 - modifying attributes 136
 - referencing 120
 - removing from folders 151
 - representation in the data model 189
 - retrieving in CM 8 141
 - retrieving linked items 156
 - searching for in CM 8 140
 - setting and retrieving attributes in CM 8 132
 - tree constant 153
 - versioning 112, 145
- ItemSetSysAttr 115

- ItemSetUserAttr 115
- ItemSQLSelect 115
- ITEMTREE 470
- ItemTypeQuery 115
- iterators
 - federated 80
 - positioning in DKResults 81

J

- j2ee.jar 405
- Java
 - constants 22
 - creating workflow actions 394
 - DKConstant.h 13
 - document viewer toolkit 529
 - error message property files 382
 - escape sequences 211
 - FeServerDefBase 382
 - garbage collector 13
 - increasing JVM stack size 337
 - packages 13
 - setting up the environment 15
 - XML functions 13
- Java Database Connectivity (JDBC) 360
- Java Server Pages (JSPs)
 - servlet 554
- java.lang.exception 118
- java.lang.objects 132
- JavaBeans,
 - See* beans
- JAX-RPC based client toolkit 520
- JDBCDRIVER, relational databases 362
- JDBCSEVERFILE, relational databases 362
- JDBCSEVERURL, relational databases 362
- JNI tracing, servlet 557
- joining routes 222
- JSPs,
 - See* Java Server Pages (JSPs)

K

- keywords, Domino.Doc 356

L

- LANG 279
- large object (LOB) 120
 - handling in earlier CM 261
- LeaveUserDLL 496
- LeaveUserFunction 496
- length, BLOB 377
- LETTER 217
- library
 - C++ listing 17
 - Java 15, 17
 - Microsoft Visual Studio .NET 19
- library server, Content Manager Version 8 107
 - listing 121
 - versioning 112
- LIKE 207, 216
- links
 - attribute type 120

links (*continued*)

- Content Manager Version 8 110
- creating in XML 490
- defining between items 154
- description item 154
- inbound and outbound 155
- representation in the data model 189
- retrieving linked items 156
- source 154
- target 154
- traversing through query 200
- type name constants 155
- type names 155
- LinkTypeName 154
- ListActionNamesRequest 501
- ListConstructor 218
- ListContent 218
- listDataSourceNames 371
- listDataSources 121, 371
- listEntityNames 124
- listFunctions
 - dkDatastoreExt 380
- listMappingNames 372
- ListNextWorkPackagesRequest 499
- ListProcessNamesRequest 497
- ListProcessRequest 497
- listResourceMgrs 121
- ListSchemaRequest 466
- ListServerRequest 464
- listWorkFlowTemplates 394
- ListWorkListNamesRequest 498
- listWorkLists 390
- ListWorkListsRequest 498
- ListWorkNodesRequest 494
- ListWorkPackagesRequest 499
- literal, query 218
- literals
 - ImagePlus for OS/390 346
- literals, query 204
- LoadFolderTSQBICDL 282
- LOCAL 443
- locale
 - beans, changing in 412
 - servlet 554
- LocalPart 218
- logging,
 - See* tracing
- LogicalOrSetExpression 217
- LogicalOrSetPrimitive 217
- LoginData 465
- LorgicalOrSetTerm 217

M

- match highlighting,
 - See* highlighting, text search
- MATCH_DICT 91
- MATCH_INFO 91
- maxResults 470
- media object
 - adding in earlier CM 51
 - code sample names 66
 - deleting 56
 - retrieving 60
- member
 - adding 268
 - removing 268

- messages
 - DKException information 117
 - property files 382
- Microsoft SOAP Toolkit 518
- Microsoft Visual C++ compiler,
 - See* Visual C++ compiler
- Microsoft Visual Studio .NET
 - getting started in Web Services 518
 - linking to 19
 - working with 19
- MIME types 526
 - setting for a resource item 140
- Mime2App property 422
- MimeMultipart object 526
- MOD 216
- Model View Controller (MVC) 406, 544
- MoveItemRequest 492
- moveObject 372
- moveObject, dkDatastoreExt 381
- multimedia content 3
- multiple characters (MC) 279
- Multipurpose Internet Mail Extensions (MIME) 526
- multistreaming 13

N

- nameOfAttrStr 136
- NameText 218
- NATIVECONNECTSTRING, relational
 - databases 361
- NewValues 492
- newVersion 481
- NoAccessACL 116
- NodeGenerator 218
- non-display names 412
- non-resource items
 - Content Manager Version 8 109
- Non-visual beans 403
- NONZERO 217
- NoPrivSet 114
- NOT 216
- note
 - constant 132
 - semantic type 132
- notelog 552
- NotifyState 501
- NotifyTime 501
- NULL, query 217

O

- object management
 - Java 263
 - creating 129, 263
 - deleting 144, 270
 - updating 137, 267
- Object Management Group (OMG) 1
- objects
 - beans 552
 - Content Manager Version 8 110
 - managing in Domino.Doc 356
 - storing attribute values 132
 - text searching for contents 192
 - updating in CM 8 138

- OnDemand
 - annotations 338
 - connecting to 327
 - disconnecting from 327
 - displaying attributes 333
 - displaying documents 333
 - enabling the folder mode 337
 - introduction 326
 - listing application groups 328
 - listing folders 330
 - listing server information 328
 - mapping from federated 6
 - property names 327
 - querying an application group 331
 - representing servers and
 - documents 327
 - retrieving documents 330
 - searching asynchronously 337
 - searching for a document 330
 - tracing 338
 - using folders as search template 338
- Open Database Connectivity (ODBC) 360
 - configuration strings 38
- open document management API (ODMA) 355
- open, BLOB 378
- open, CLOB 378
- open, result set cursor 376
- operators
 - Domino.Doc 360
 - ImagePlus for OS/390 query 346
 - parametric search 190
- OptionalExpressionList 218
- OptionalPredicateList 218
- OR 216
- OracleText
 - query example 201
- outbound links 153, 155
- output_option 25
- overlay object 381
- OverLoadLimit 494
- OverloadUserDLL 496
- OverloadUserFunction 496

P

- packages
 - client and server (cs) 14
 - hierarchy in Java 14
 - Java client 13
 - Java server 13
- page
 - manipulation
 - functions 546
- page manipulation
 - interfaces 546
- pages
 - beans 554
- parallel routing 222
- parametric search
 - definition 20
 - evaluating a query from a content
 - server 87
 - federated 9
 - formulating a query string 82
 - formulating multiple criteria 83
- parametric search (*continued*)
 - operators 190
 - querying from a content server 85
 - querying in CM for AS/400 354
 - querying on multiple criteria 83
 - running the query 84
 - tracing 21
 - understanding 190
- parts
 - earlier CM 262
 - updating in earlier CM 267
 - versioning 180
- password
 - changing 371
 - changing in beans 424
 - changing in XML 466
 - creating in Content Manager Version 8 122
 - mapping in federated 7
- pattern 298
- PENDING, ImagePlus for OS/390 347
- pEnt, C++ 28
- permissions,
 - See* privileges
- Persistent Data Service (PDS), COBRA 1
- persistent identifier (PID)
 - dynamic data object (DDO) 2
 - extended data object (XDO) 2
 - image search 307
 - introduction 4
 - routing 242
 - versioning 112
- Persistent Object Service, COBRA 1
- policies example 119
- positional color
 - definition 297
 - query example 301
 - valid values 301
- predefined values 136
- predicate list 218
- priority, document routing 501
- privileges
 - access control lists (ACLs) 115
 - beans 408, 553
 - codes 115
 - Content Manager Version 8 113
 - creating a set 158
 - creating in CM 8 157
 - data access 114
 - displaying properties of a set 160
 - granting for routing 246
 - pre-configured ACLs 116
 - pre-defined Content Manager Version 8 sets 114
 - set 114
 - system-defined 114
 - types of sets 114
 - users and user groups in CM 8 115
 - viewing in XML 476
- PrivSetCode 114
- process image queue 311
- process state 225
- ProcessCompletionTime 501
- processes
 - continuing 240
 - continuing in XML 506
 - defining for an associated route 236

- processes (*continued*)
 - DKProcessICM 224
 - ending 240
 - joining 222
 - listing 244
 - listing in XML 497
 - resuming 241
 - resuming in XML 508
 - routing 224
 - splitting 222
 - starting in XML 505
 - subprocesses 224
 - suspending 241, 507
- profiles, Domino.Doc 356
- PublicReadACL 116

Q

- QbColor 300
- QbColorFeatureClass 297, 300
- QbColorHistogramFeatureClass 297, 300, 301
- QbDraw 301
- QbDrawFeatureClass 297, 301
- QbHistogram 300
- QbTexture 301
- QbTextureFeatureClass 298, 301
- QName 218
- queries
 - arithmetic operations 200
 - average color search 301
 - CM for AS/400 349
 - combined expressions 370
 - compound expressions 370
 - Content Manager Version 8 187
 - data model 197
 - dkQuery class 370
 - Domino.Doc 359
 - Domino.Doc syntax 359
 - escape sequences 206
 - evaluate 82
 - examples 196, 199
 - execute 82
 - federated
 - processing 8
 - federated collection 80
 - image search 299, 307
 - image search query syntax 299
 - ImagePlus syntax 345
 - language 206
 - language grammar 216
 - listing checked out items 205
 - listing the result 205
 - OnDemand search 330
 - relational databases 365
 - routing examples 246
 - searching structured documents 285
 - syntax 199
 - text search 192
 - understanding the language 187
 - version 203
 - wildcard example 204
 - XML items 469

- Query
 - Java 81
 - dkResultSetCursor vs DKResults 82

- Query (*continued*)
 - Java (*continued*)
 - parametric type 82
 - query object types 81
 - text type 88
- Query by Image Content (QBIC), *See* image search
- query language 206
- Queryable collection
 - Java 103
 - evaluating 104
 - getting results 103
 - Programming tips 105
 - queryable vs refined 105
- QueryCriteria 470

R

- reference attributes 190
- REFERENCEDBY 190
- REFERENCER 190
- references
 - attribute type 120
 - Content Manager Version 8 111
 - traversing through query 202
- registerMapping 372
- registerServices 371
- relational databases
 - configuration strings 361
 - connecting to 360
 - connection strings 361
 - introduction 360
 - listing attributes 362
 - listing entities 362
 - mapping from federated 6
 - querying 365
- relative positioning, result set cursor 102
- remote method invocation (RMI)
 - connecting with beans 406
 - starting in Java 16
 - using with Java APIs 16
- Remote Procedure Call (RPC) 514
- remove, BLOB
 - binary large object (BLOB)
 - removing a portion 378
- remove, CLOB 379
- removeAllElement 79
- removeFromFolder 380
- RemoveItemFromFolderRequest 478
- removeMember 268
- replaceElementAt 79
- ReplayAction, servlet 555
- RepType 282
- resource content, *See* XDO
- resource items
 - constant 122
 - Content Manager Version 8 109
 - defining an item type for 138
 - item type 122
 - representing in Web services 526
- resource manager, Content Manager
 - Version 8 107
 - listing 121
 - versioning 112
 - working with 167
 - working with objects 167

- result set cursor
 - functions 375
- Result set cursor
 - Java 99
 - creating a collection 102
 - open and close 100
 - set and get 100
- resume list 227
- ResumeList 501
- ResumeListDefinition 508
- ResumeProcessRequest 508
- ResumeTime 501
- resuming a workflow 390
- Retrieval
 - C++
 - parts 272
 - Java 271
 - folders 273
 - parts 271
- retrieve, BLOB 377
- RetrieveFolderItemsRequest 478
- retrieveFromOverlay 381
- RetrieveItemRequest 472
- retrieveObject 371, 372
- retrieveOption 470
- RMI server, *See* remote method invocation (RMI)
- rollback 371
- rooms, Domino.Doc 356
- root components
 - Content Manager Version 8 109
 - creating in CM 8 130
 - linking 110
 - referencing 111
 - representation in the data model 189
 - versioning 112
- RouteSelected 506
- routing 410
 - access control lists (ACLs) 247
 - ad hoc 245
 - beans 407, 410
 - collection points 221
 - compatibility with Version 8.2 224
 - constants 225, 251
 - container variables 223
 - continuing a process 240
 - creating service objects 227
 - decision point 222
 - decisions 225
 - defining a new collection point 231
 - defining a new process and associated route 236
 - defining a new regular work node 228
 - defining a worklist 234
 - definition 225
 - diagram definition 225
 - ending a process 240
 - examples 246
 - granting privileges 246
 - graphical process builder 222
 - introduction 221
 - joining 222
 - listing document routing processes 244
 - listing package PIDs in a worklist 242

- routing (*continued*)
 - listing work nodes 230
 - listing worklists 235
 - parallel 222
 - process state 225
 - resume list 227
 - resuming a process 241
 - retrieving work package information 243
 - setting up 224
 - splitting 222
 - starting 239
 - starting in XML 504
 - subprocesses 224
 - suspending a process 241
 - user exits 248
 - Version 8.3 APIs 222
 - virtual nodes 225
 - work nodes 221
 - work package 227
 - workflow action 223
 - worklist 227
 - XML 493
- RunQueryRequest 469

S

- sample client 295
- sample files
 - Content Manager Version 8 118
- SampleMessageTemplate.java 522
- SAttributeDefinitionCreationICM 109
- scenario, insurance sample for Content Manager Version 8 119
- schema mapping
 - associating in federated 6
 - custom connectors 369
 - dkDatastore methods 372
 - dkSchemaMapping class 370
 - introduction 1
 - relational databases 361
- schemas
 - beans 550
 - cmadmin.xsd 429
 - cmdatamodel.xsd 429
 - conversion tool 458
 - importing and exporting storage schemas 434
 - unsupported XML types in storage schemas 453
- SConnectDisconnectICM 118
- scope
 - tag library 550
 - transaction 181
- score-basic, text search 192, 193
- scrolling the result set cursor 375
- SDocModelItemICM 111
- SDocRoutingDefinitionCreationICM 228, 229, 234, 235, 239
- SDocRoutingListingICM 231, 236, 243, 245
- SDocRoutingProcessingICM 239, 240, 241, 242, 244
- search engines, registering 371
- search template, OnDemand
 - using folders as 338
 - viewer 337
- searching
 - API modules 119
 - beans 408, 550, 552
 - DKResults class 370
 - understanding the query language 187
- searchTemplate bean 551
- SelectionFilterOnNotify 498
- SelectionFilterOnOwner 499
- SelectionFilterOnSuspend 499
- SelectionOrder 498
- semantic type
 - definition 131
 - pre-defined types 131
 - user defined 132
- SequencedValue 217
- ServerType 464
- servlet,
 - See* controller servlet
- session listeners, beans 411
- sessions
 - servlet 554
- setClassOpenHandler 378, 379
- setContent 377
- setContentFromClientFile 377, 379
- setData 131
- setDatastore 380
- setInstanceOpenHandler 378, 379
- setLocale 412
- setPosition 375
- settings
 - C++ environment 17
 - Java environment 15
- Settings
 - C++
 - Building on NT 18
 - Java
 - Client connect and disconnect 24
 - Client/Server 13
 - On NT 15, 18
 - Programming tips 14
 - Using sample Java applets and servlet 549
 - Java application on client 549
 - Local access 550
 - Remote access 550
 - Retrieve servlet 549
- setToBeIndexed 311
- setToFirstCollection 81
- setToLastCollection 81
- setToNext 376
- setToNextCollection 81
- setToPreviousCollection 81
- SFolderICM 111, 148, 154
- Shortcut 503
- Simple Object Access Protocol (SOAP) 513
- single required characters (SC) 279
- SItemCreationICM 111
- SItemDeletionICM 144
- SItemRetrievalICM 144
- SItemTypeCreationICM 109
- SItemTypeRetrievalICM 147
- SItemUpdateICM 140, 145
- SLinksICM 111, 155, 156
- SLinkTypeDefinitionCreationICM 156
- SORTBY 216, 217

- SortSpec 217
- SortSpecList 217
- SOURCEITEMREF 190
- splitting routes 222
- SReferenceAttrDefCreationICM 111, 190
- SResourceItemMimeTypeICM 140
- SSearchICM 141
- stack trace 117
- StartProcessRequest 504
- Step, query 218
- storage collection
 - adding an XDO to 65
 - changing for an XDO 67
- storage schemas
 - importing and exporting 434
 - unsupported XML types 453
- streaming doc services 530
- STRING_LITERAL 217
- subentities
 - listing in Domino.Doc 357
- subprocesses 224
- subString 378, 379
- SuperUserACL 116
- suspending a process 241
- suspending a workflow 389
- SuspendProcessRequest 507
- SuspendState 501
- svWF 388
- SYSREFERENCEATTRS 190
- system administration client
 - creating worklists 391
 - customizing 12
 - user exits 12
- System Managed Storage (SMS), Content Manager Version 8 107
- SystemAdmin 115
- SystemAdminPrivSet 114
- SystemDefineItemType 115

T

- tag library
 - connection related 550
 - document related 553
 - folder related 553
 - item related 552
 - schema related 551
 - search related 552
- tagged documents
 - text search 285
- taglib.tld 405
- TARGETITEMREF 200
- TCallbackOD 337
- TCheckStatusTS 294
- TCP/IP
 - text search 278
- templates
 - workflow 394
- TerminateProcessRequest 505
- text information extender (TIE)
 - Net Search Engine (NSE) 20
- text search
 - advanced search 193
 - basic query 192
 - beans 417
 - contains-text 193
 - creating the document model 287

- text search (*continued*)
 - definition 20
 - evaluating a query from a content server 91
 - formulating a query string 88
 - highlighting 91
 - listing the document model 285
 - making user-defined attributes searchable 192
 - management functions 274
 - OnDemand 326
 - operators 192
 - persistent identifier (PID) 277
 - query example 201
 - query syntax 192
 - queryable collections 104
 - querying from a content server 90
 - querying multiple indexes 88
 - querying on multiple indexes 88
 - running a query 88
 - score-basic 193
 - searching for documents 192
 - searching for object contents 192
 - searching structured documents 285, 294
 - setting indexing rules 289
 - starting the indexing process 292
 - tracing 20
 - understanding 191
 - wildcards 193
- texture
 - coarseness 298
 - contrast 298
 - definition 298
 - directionality 298
 - query example 301
 - valid values 301
- thin client 533
- TImageAnnotation 546
- timed-out sessions, servlet 554
- TimeLastMoved 501
- TimeLimit 494
- tools
 - Content Manager Version 8 118
- toXML() 454
- tracing
 - beans 413, 550
 - controller servlet 556
 - error state 117
 - handling errors with
 - DKException 117
 - JNI, servlet 557
 - OnDemand 338
 - parametric search 21
 - return code container 117
 - sample tools 118
 - stack 117
 - text search 20
- transactions, Content Manager Version 8 181
 - caution about explicit transactions 182
 - checking in and out 183
 - considerations 182
 - list 184
 - processing 183
- transformXMLDocument 460

- TRetrieveFolderWithCallbackOD 337
- TRetrieveWithCallbackOD 337
- troubleshooting
 - error message property files 382
 - tracing 20
- TxdoAdd samples 66
- TxdoAsyncRetDL 261

U

- UNICODE_CHARACTER 217
- uniform resource identifier (URI) 168
- UNION 206, 216
- unlockCheckedOut 381
- unRegisterMapping 372
- unregisterServices 371
- update, BLOB 377
- UpdateFTIndex 359
- UpdateItemRequest 480
- updateObject 138, 372
- UpdateWorkPackageRequest
 - XML messages
 - updating work packages 503
- updating content 379
 - XDO 45
- user exits
 - programming for document routing 248
 - system administration 12
 - workflow 394
- user ID
 - mapping in federated 7
- user privileges
 - Content Manager Version 8 113
- user-defined attributes 189
 - making text searchable 192
- UserLastMoved 501
- users
 - controlling access in CM 8 115

V

- valueObj 136
- version attribute 470
- version, querying 203
- versioning
 - application-controlled 112
 - attributes 143
 - beans 414, 423
 - Content Manager Version 8 112
 - item types 147
 - items 142
 - parts in the document management
 - data model 180
 - persistent identifier (PID) 112
 - policy 112
 - setting for an item 145
 - version control policies 146
- VideoCharger 107
- VIEW 443
- viewers, Java document,
 - See* document viewer toolkit
- virtual nodes 225
- Visual C++ compiler 17
- Visual Studio .NET
 - libraries 19

- Visual Studio .NET (*continued*)
 - overview of WSDL 516

W

- W3C XML Query 187
- wakeUpService 284
- Web services
 - advantages 514
 - architecture 515
 - authenticating 523
 - C# walkthrough sample 519
 - creating a new item instance 524
 - development tools 516
 - getting started in .NET 518
 - getting started in a Java
 - environment 520
 - introduction 513
 - Java samples 518
 - Java template listing 522
 - Java walkthrough sample 521
 - overview 513
 - programming in .NET 519
 - relation to other XML services 429
 - samples location 517
 - scenario 513
 - security 523
 - standards 514
 - walkthrough sample 519
- Web Services Description Language (WSDL) 514
- web.xml 405
- WebSphere Application Server
 - authenticating 465
- WebSphere Studio Application Developer (WSAD)
 - building beans 404
 - creating a Web services client
 - with 519
 - diagram 431
- wildcards
 - Domino.Doc 360
 - query 207
 - text search 193
- Windows
 - C++ DLL files 17
 - setting the console subsystem 19
 - starting the RMI server 16
- work items
 - accessing 392
- work nodes
 - business application 223
 - constants 225
 - decision points 225
 - DKWorkNodeContainerDefICM 226
 - extending 496
 - listing 230
 - listing in XML 494
 - types 496
 - virtual 225
- work note
 - defining 228
 - routing 221
- work packages
 - DKWorkPackageICM 227
 - listing in XML 499
 - listing PID strings in a worklist 242

- work packages (*continued*)
 - listing the next one in XML 499
 - retrieving information 243
 - updating in XML 503
 - user exits 248
- work packet
 - introduction 385
- workbasket
 - creating in earlier CM 321
 - definition in earlier CM 318
 - identifying in earlier CM 325
 - listing in earlier CM 322
 - rules 318
- workflow
 - accessing work items 392
 - accessing worklists 391
 - actions 223
 - beans 405
 - beans support 407
 - connecting to 385
 - container variables 223
 - creating in earlier CM 319
 - creating Java actions 394
 - definition in earlier CM 318
 - deleting 387
 - disconnecting from 385
 - earlier Content Manager service 318
 - executing in earlier CM 325
 - introduction 385
 - listing 388
 - listing all the templates 394
 - listing all the worklists 390
 - listing in earlier CM 320
 - listing items IDs in earlier CM 323
 - model 318
 - moving items 393
 - sending results from a federated
 - folder 11
 - starting 386
 - suspending 389
 - terminating 387
- WorkListName 499
- worklists
 - accessing 391
 - accessing work items 392
 - beans 405
 - defining 234
 - DKWorkListICM 227
 - listing 235, 390
 - listing in XML 498
 - listing package PIDs 242
 - moving items 393
- WorkPackageOwner 499
- WorkPackagesToReturn 499
- wsdl.exe
 - getting started with 519
 - overview 516
- WSDL2Java 520

X

- XDO
 - adding a media object 51
 - adding an annotation object to 43
 - adding from a file 42
 - adding from the buffer 41
 - adding to a storage collection 65

- XDO (*continued*)
 - changing the storage collection 67
 - class 369
 - class type hierarchy 139
 - comparing with attributes 3
 - creating a resource item 139
 - deleting 45, 56
 - dkBlob 377
 - dkXDO 369
 - examples 45
 - indexing in earlier CM 38
 - indexing in image search 311
 - introduction 2
 - invoking a function 46
- Java 37
 - data properties 37
 - DDO, part of 39
 - indexing 281
 - PID 37
 - Programming tips 38, 39
 - stand-alone 41
- large object 120
- members 378
- objects 110
- representing multimedia content 3
- retrieving 45
- retrieving a media object 60
- setting the MIME type 140
- specific DK classes 381
- text searching 202
- updating 45

- XML
 - administration metadata 433
 - beans 408
 - beans tag library 405
 - document routing 493
 - extracting
 - buffer 458
 - file 458
 - Web address 458
 - item types 434
 - items 454
 - Java support 13
 - representation of the query examples
 - data model 197
 - representing metadata 455
 - schema conversion tool 458
 - services 429
 - W3C query 187
- XML messages
 - authenticating 465
 - batching 509
 - changing a password 466
 - checking items out and in 488
 - continuing a process 506
 - creating an item 468
 - deleting items 487
 - ending a process 505
 - linking items 490
 - listing actions 501
 - listing entities 466
 - listing processes 497
 - listing servers 464
 - listing work nodes 494
 - listing work packages 499
 - listing worklists 498
 - moving items between entities 492

- XML messages (*continued*)
 - relation to Web services 515
 - resuming a process 508
 - retrieving items 472
 - searching items 469
 - SOAP 513
 - starting a process 504
 - suspending a process 507
 - updating items 480
 - viewing user privileges 476
 - working with folders 477
- XML services
 - diagram 430
 - exporting object dependencies 457
 - importing and exporting
 - metadata 432
 - overview 429
 - programming layers 429
 - programming with JavaBeans 461
 - scenarios 432
 - unsupported XML types 453
- XMLItem 455
- XQuery Path Expressions (XQPE) 187
- XSD files 434



Program Number: 5724-B19
5697-H60

Printed in USA

SC27-1347-04



This document was filed with PUCO Docketing on

7/18/2006 @ 3:19:01 PM