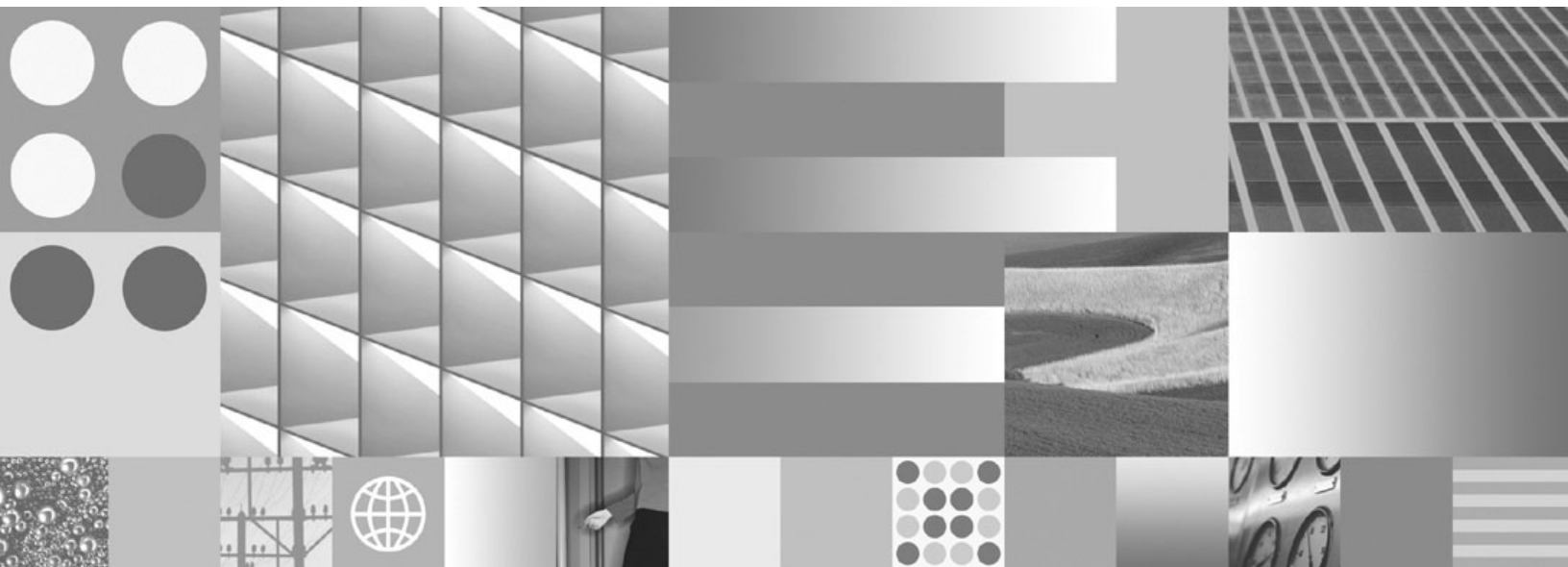
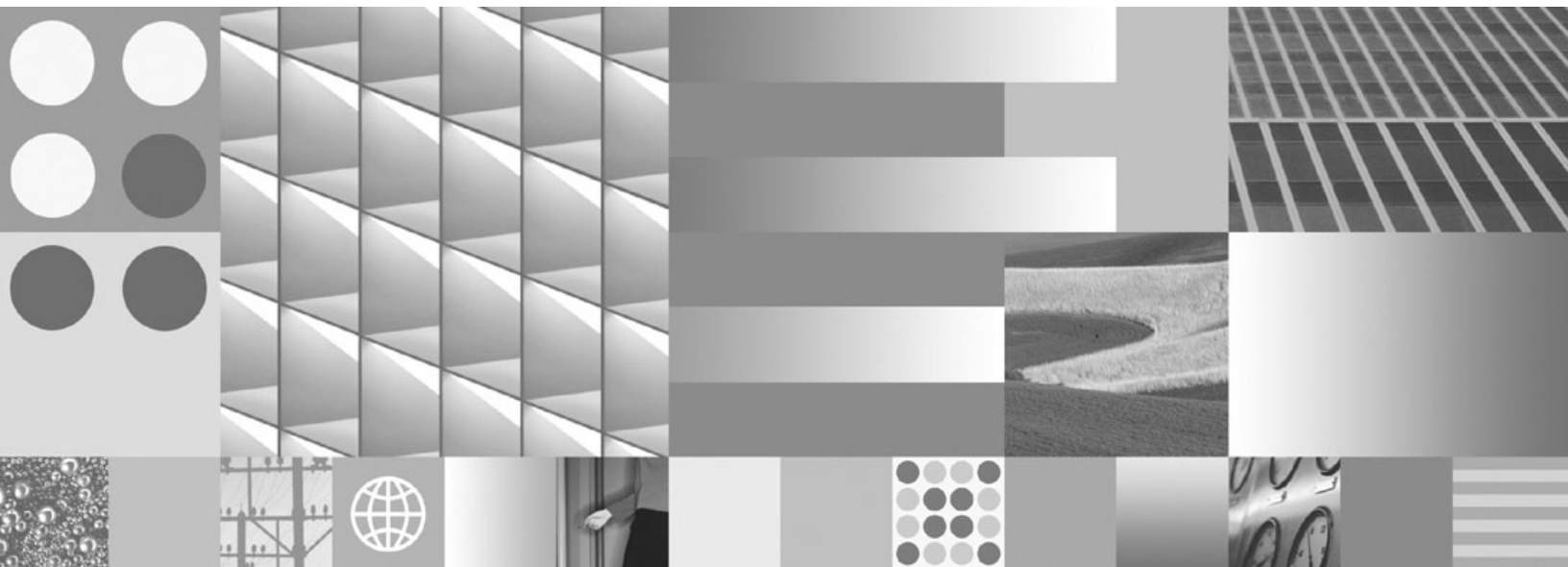




Application Programming Guide



Application Programming Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 501.

This edition applies to version 8, release 4 of IBM DB2 Content Manager Enterprise Edition (product number 5724-B19) and version 8, release 4 of IBM DB2 Content Manager for z/OS (product number 5697-H60) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1996, 2007. All rights reserved.

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

Contents

ibm.com and related resources ix

How to send your comments ix

Contacting IBM x

About this guide 1

Who should use this guide 1

Getting started with programming content management applications . . . 3

Important information for Linux users. 4

Deprecated and removed features 4

Data access through content servers 5

Dynamic data object concepts. 6

Dynamic data objects (DDO) 6

Extended data objects (XDO) 7

Representing multimedia content 7

Content servers and DDOs 7

Comparing DDO/XDOs with attribute values and
item parts 8

Persistent identifiers (PID) 8

Working with a federated content server and federated searching (deprecated) . . 9

Federated schema mapping 10

Using federated content server mapping
components 10

Running federated queries 11

Federated query syntax 12

Storing query results in federated folders (Java
only) 14

Working with system administration 15

Customizing the DB2 Information Integrator for

Content system administration client 15

Programming with the application programming interfaces (APIs) 17

Understanding differences between the Java and

C++ APIs 17

Understanding client/server architecture (Java only) 17

Packaging for the Java environment 18

Programming tips 18

Setting up the Java environment (Java only) . . . 19

Setting the Java environment variables for

Windows 19

Setting the Java environment variables for AIX 19

Setting the Java environment variables for Solaris
and Linux 20

Setting the Java environment variables for z/OS

USS 20

Using Remote Method Invocation (RMI) with
content servers 20

Setting up the C++ environment (C++ only) . . . 21

Setting the C++ environment variables for

Windows 22

Setting the C++ environment variables for AIX 22

Building C++ programs 23

Setting the console subsystem for code page
conversion on Windows 23

Java and C++ application performance 23

Java API global caching 23

Native API renovated retrieve options 24

C++ API scalability and performance: global

caching and connection pooling 25

Timing out connections with C++ API 27

Indicating elements fetched when retrieving the
list of ACLs 28

Understanding multiple search options 29

Handling exceptions 29

Constants 31

Connecting to content servers 31

Establishing a connection. 32

Connecting and disconnecting from a content
server in a client. 32

Setting and getting content server options . . . 33

Listing content servers. 33

Listing the entities and attributes for a content
server 34

Working with dynamic data objects (DDOs) . . . 36

Creating a DKDDO. 36

Adding properties to a DDO 37

Creating a persistent identifier (PID) 37

Working with data items and properties. 38

Getting the DKDDO and attribute properties . . 40

Displaying the whole DDO 41

Deleting a DDO (C++ only) 42

Working with extended data objects (XDOs) . . . 42

Using an XDO persistent identifier (PID) . . . 43

Java programming tips 43

Programming an XDO as a part of DDO 44

Programming a stand-alone XDO 45

Examples of working with an XDO 48

Creating documents and using the DKPARTS
attribute 49

Creating folders and using the DKFOLDER attribute 52

Using DKAny (C++ only) 54

Using type code 55

Managing memory in DKAny 55

Using constructors 55

Getting the type code 55

Assigning a new value to DKAny 55

Assigning a value from DKAny 56

Displaying DKAny 56

Destroying DKAny 56

Programming tips 57

Using collections and iterators 57

Using sequential collection methods 57

Using the sequential iterator. 58

Managing memory in collections (C++ only) . . 59

Understanding federated collection and iterator 60

Querying a content server 61

Differences between dkResultSetCursor and DKResults	62	Storing content in a user-specified collection by using Java or C++ API	153
Using parametric queries	62	API for resource manager in Version 8.4	154
Working with DB2 Content Manager Version 8.4	63	Managing documents in DB2 Content Manager	158
Understanding the DB2 Content Manager system.	63	Creating the document management data model	159
Understanding DB2 Content Manager concepts	64	Creating a document item type	159
Items	64	Creating a document	161
Attributes	64	Updating a document	163
Item types	65	Retrieving and deleting a document	164
Root and child components	65	Versioning of parts in the document management data model	165
Objects	66	Working with transactions	165
Links and references	67	Things to consider when designing transactions in your application	166
Enabling auto-linking with version-enabled item types	67	Caution when using transactions	167
Documents	68	Using check-in and check-out in transactions	167
Folders	69	Processing transactions	167
Versioning	69	Explicit transactions behavior in Version 8.4	168
Access control	70	Transaction behavior when deleting a user does not belong in a group	169
Planning a DB2 Content Manager application	77	Searching for data	171
Determining the features of your application	77	Querying the DB2 Content Manager server	171
Handling errors	78	Applying the query language to the DB2 Content Manager data model	173
Enabling IPv6 dual-stack support	79	Understanding parametric search.	174
IBM DB2 driver for JDBC and SQLJ support in DB2 Content Manager Version 8.4	80	Understanding text search	175
Working with the DB2 Content Manager samples.	82	Searching for object contents	176
The insurance scenario sample	83	Searching for documents	176
Creating a DB2 Content Manager application	83	Making user-defined attributes text searchable	176
Understanding the software components	83	Understanding text search syntax.	176
Representing items using DDOs	83	Creating combined parametric and text search	183
Connecting to the DB2 Content Manager system	84	Example searches using the query language	184
Working with items	85	Query examples	188
Working with folders	113	Using escape sequences in your queries	194
Defining links between items	119	Using escape sequences with comparison operators	195
Working with access control	122	Using escape sequences with the LIKE operator	195
Creating a privilege	122	Using escape sequences with advanced text search	196
Creating a privilege set	123	Using escape sequences with basic text search (contains-text-basic and score-basic functions)	197
Displaying privilege set properties	125	Using escape sequences in Java and C++	198
Defining an access control list (ACL)	125	Understanding row-based view filtering in query	199
Retrieving and displaying ACL information	127	Important considerations for using row-based filtering	199
Assigning an ACL to an item type	127	Sample usage scenario	200
Assigning an ACL to an item	128	Description of filtering behavior	202
Using view ACLs on delete, add, or remove link operations	129	Performance considerations.	203
Implementing single sign-on in your applications	130	Database indexes for each filtered attribute	204
Library server user exits for DB2 Content Manager	132	Security implications	205
User exit ICMFetchContentExit for changing text.	133	The query language grammar	205
User exit ICMLogonExit for logon	136	Routing a document through a process	209
User exit myExit for document routing.	139	Understanding the document routing process	209
User exit ICMValidatePassword for validating passwords	140	Document routing features	210
ACL user exit routines	143	Understanding compatibility with versions prior to Version 8.3	212
Library server and federated database limitations	146		
Working with the resource manager	147		
Working with resource manager objects	148		
Confidential retrieval of resource objects	148		
Removing resource object contents	149		
Asynchronous replication on a z/OS resource manager	150		

Changing the DOCROUTINGFREQ column in the ICMSTSYSCONTROL table for DB2	212
Understanding document routing classes	213
Creating document routing service objects.	216
Defining a new regular work node	217
Listing work nodes	218
Defining a new collection point	219
Defining a worklist	221
Listing worklists	223
Defining a new process and associated route	223
Starting a document routing process.	225
Ending a process	226
Continuing a process.	226
Suspending a process.	227
Resuming a process	228
Listing work package persistent identifier strings in a worklist	228
Retrieving work package information	229
Listing document routing processes	230
Implementing ad hoc routing	231
Document routing example queries	232
Granting privileges for document routing	232
Working with access control lists for document routing	234
Programming document routing user exits	234
Document routing constants	235

Understanding prefetching in DB2

Content Manager for z/OS 237

Prefetching objects.	237
Table definitions related to prefetching	239

Working with other content servers 243

Working with OnDemand	244
Representing OnDemand servers and documents	244
Connecting to and disconnecting from the OnDemand server.	245
Listing information on OnDemand	245
Retrieving an OnDemand document.	248
Enabling the OnDemand folder mode	253
Asynchronous search.	253
OnDemand folders as search templates.	254
OnDemand folders as native entities	254
Create and modify annotations	254
Tracing	254
Working with ImagePlus for OS/390	256
Listing entities and attributes	256
ImagePlus for OS/390 query syntax.	261
Working with DB2 Content Manager for AS/400	262
Listing entities (index classes) and attributes	262
Running a query	264
Running a parametric query	268

Building DB2 Information Integrator for Content workflow applications (deprecated). 271

Connecting to workflow services	271
Starting a workflow	272
Terminating a workflow.	273

Listing all the workflows	273
Suspending a workflow	274
Resuming a workflow	275
Listing all the worklists	275
Accessing a worklist	276
Accessing work items	277
Moving items in the workflow	278
Listing all the workflow templates	278
Creating your own actions (Java only)	279
Working with the DB2 Information Integrator for Content workflow JavaBeans	280
Prerequisites.	280
Setting up the sample data model	280
Using workflow JavaBeans in your application	287

Building applications with non-visual and visual JavaBeans. 291

Understanding basic beans concepts.	291
Using JavaBeans in builders	292
Using IBM Websphere Studio Application Developer	293
Invoking the DB2 Information Integrator for Content JavaBeans.	293
Working with the non-visual beans	294
Non-visual bean configurations	294
Understanding the non-visual beans features	295
Non-visual beans categories	295
Considerations when using the non-visual beans	300
The CMBConnectionPool bean	301
Display names	304
Tracing and logging in the beans.	305
Understanding properties and events for non-visual beans	305
Building an application using non-visual beans	306
Working with visual beans	306
CMBLogonPanel bean	306
CMBSearchTemplateList bean	308
CMBSearchTemplateViewer bean.	309
Validating or editing fields of the CMBSearchTemplateViewer.	310
CMBSearchPanel bean	310
CMBSearchResultsViewer bean	310
Overriding pop-up menus	312
CMBFolderViewer bean	312
CMBDocumentViewer bean	313
Viewer specifications	314
Default viewers	315
Launching external viewers	315
CMBItemAttributesEditor bean	315
Vetoing changes in the CMBItemAttributesEditor	316
CMBVersionsViewer bean	316
General behaviors for visual beans	316
Replacing a visual bean	318
Building an application using visual beans	318

Working with XML services (Java only) 321

How XML services work with other DB2 Content Manager programming layers	321

Importing and exporting DB2 Content Manager metadata using XML services	324	Installation and configuration of Web services	411
Importing and exporting administration objects as XML	325	Supported WebSphere Application Server topologies	411
Importing and exporting DB2 Content Manager data model objects as XML schema files (XSD)	326	Verifying, starting, and stopping the Web services	413
Importing and exporting DB2 Content Manager data instance objects as XML	345	Post installation configuration	413
Exporting DB2 Content Manager DDO items as XML items	346	Logging and tracing	415
Importing XML items as DB2 Content Manager DDO items	347	Un-installing and removing configuration	416
Importing and exporting XML object dependencies	349	Integrating Web services into your applications or processes	417
Extracting content from different XML sources	349	Integrating CMWebService	417
Mapping a user-defined schema to a storage schema with the XML schema mapping tool	350	Integrating CMBGenericWebService	425
Programming run-time operations through the XML JavaBeans.	352	Authenticating Web services requests for security	428
Listing DB2 Content Manager servers with ListServerRequest	354	Creating a new instance of an item through Web services	429
Authenticating Web service requests for security	355	Configuring single sign-on	434
Changing a password with XML requests	356	Web services performance	436
Listing DB2 Content Manager entities with ListSchemaRequest	357	Web services samples.	437
Creating DB2 Content Manager items with CreateItemRequest	358	Web services troubleshooting	437
Searching DB2 Content Manager items with RunQueryRequest	360	Setting the cached statement in WebSphere Application Server connection pool for Web services	437
Retrieving DB2 Content Manager items with RetrieveItemRequest	362	Using the forceItemNamespace property when developing client applications that use the Web services	438
Viewing your user privileges with XML requests	367	Update Web services server configuration to refer to the 32-bit libraries when using a DB2 64-bit instance	438
Working with DB2 Content Manager folders through XML requests	368		
Updating DB2 Content Manager items with an XML UpdateItemRequest	371	Working with the Java document viewer toolkit	441
Deleting DB2 Content Manager items with DeleteItemRequest.	377	Java viewer toolkit overview	441
Checking DB2 Content Manager items out and in with CheckoutItemRequest and CheckinItemRequest	378	Viewer architecture	443
Linking DB2 Content Manager items with CreateLinkRequest or DeleteLinkRequest	380	The document engines	443
Moving DB2 Content Manager items between entities with MoveItemRequest	381	The annotations engine	444
Accessing DB2 Content Manager document routing using XML-based requests	383	Example viewer architectures	444
Batching multiple requests in XML requests	399	Creating a document viewer	447
		Creating a standalone viewer application or applet	447
Working with the Web services.	401	Working with documents and annotations.	449
Web services overview	401	Customizing the viewer	450
DB2 Content Manager Web services	402	Working with the annotation services	455
DB2 Content Manager Web services implementation.	403	Using annotation services interfaces	456
DB2 Content Manager Web services architecture	403	Understanding annotation editing support	457
Comparison of the DB2 Content Manager Web services interfaces	404	Building an application using the annotation services	458
Web services functions	406	Adding a custom annotation type to your application	458
List of Web services requests	407	Supported document types and conversions in Java viewer toolkit	459
Installing and configuring the DB2 Content Manager Web services	410	Working with the page manipulation functions	461
Platforms supported by the Web services	410	Engine property settings for compression	464
		Specifying character encoding information for plain text documents.	464
		Modifying viewer preferences	465
		Java document viewer limitations	472
		Working with the JSP tag library and controller servlet	473

Setting up the tag library and servlet	473	Cannot add, store, retrieve, or update a resource item	492
Using the tag library	473	Cannot import a DKDDO object from XML	493
Conventions used in the tag library	474	Receiving an error when updating, reorganizing, or using text indexes for text searchable components	493
Tag summary	474	Cannot download ICMBASESTREAM part type content	494
Connection related tags	474	Text search does not return results for documents created in certain languages	494
Schema related tags	475	DKString DKLobICM::getContent () const not recommended for ICM connector.	495
Search related tags	476	Cannot load user exit.	495
Item related tags	476	Cannot call some administration APIs in an explicit transaction	496
Folder related tags.	477	Connect() API does not check for null or empty password in Trusted Logon mode	497
Document related tags	478	Wildcard character search of the library server in z/OS can sometimes return incorrect data.	497
DB2 Information Integrator for Content controller servlet.	478	Receiving a DKException message when query times out.	498
What the servlet can do	478	Java viewer does not support Excel documents that have document protection enabled	498
Servlet reference	479	Java viewer: UTF-8 and UTF-16 plain text documents always appear left-aligned	499
Servlet toolkit function matrix.	484	Java viewer: Line wrapping causes incorrectly positioned graphical annotations relative to the text	499
 Troubleshooting content management applications	 487	 Notices	 501
Java, C++, and DB2 Information Integrator for Content connector logging	487	Trademarks	503
Java connector logging	487	 Glossary	 505
C++ connector logging	488	 Index	 515
Modifying the logging configuration file for the DB2 Information Integrator for Content connectors	488		
Receiving an error when compiling C++ applications that are Unicode enabled	490		
Receiving an error when using reference attributes	490		
Receiving an error when WebSphere Application Server connection pooling is enabled	491		
Receiving a search error in the Thai language when database setup is incorrect	491		
Receiving an error when using ICM text search for Thai phrases in DB2	492		

ibm.com and related resources

Product support and documentation are available from [ibm.com](http://www.ibm.com).

Support and assistance

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

DB2 Content Manager

<http://www.ibm.com/software/data/cm/cmgr/mp/>

DB2 Content Manager for z/OS

<http://www.ibm.com/software/data/cm/cmgr/390/>

Information center

You can view the product documentation in an Eclipse-based information center that you can install when you install the product. By default, the information center runs in a Web server mode that other Web browsers can access. You can also run it locally on your workstation. See the information center at <http://publib.boulder.ibm.com/infocenter/cmgmt/v8r4m0/index.jsp>.

PDF publications

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

See the following PDF publications Web sites:

Product	Web site
IBM DB2 Content Manager	http://www.ibm.com/support/docview.wss?rs=86&uid=swg27010186
IBM DB2 Content Manager for z/OS	http://www.ibm.com/support/docview.wss?rs=119&uid=swg27010286

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

Send your comments by using the online reader comment form at https://www14.software.ibm.com/webapp/iwm/web/signup.do?lang=en_US&source=swg-rcf.

Contacting IBM

To contact IBM customer service in the United States or Canada, call 1-800-IBM-SERV (1-800-426-7378).

To learn about available service options, call one of the following numbers:

- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

For more information about how to contact IBM, see the Contact IBM Web site at <http://www.ibm.com/contact/us/>.

About this guide

This guide describes how to use the Java™, JavaBeans™, and C++ application programming interfaces (APIs) provided with DB2 Information Integrator for Content and DB2 Content Manager Version 8.4. The APIs and beans provide building blocks for creating applications that access content stored in heterogeneous content servers.

This guide includes:

- An introduction to DB2 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 8.4 connector
- Documentation on all other DB2 Information Integrator for Content connectors to content servers
- Updates to visual and non-visual JavaBeans
- Updates to programming information for workflow

Illustrations referring to DB2 Content Manager imply both pre-Version 8.1 and Version 8.4 of the product.

Who should use this guide

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

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

Getting started with programming content management applications

DB2 Content Manager provides Java™, JavaBeans™, and C++ application programming interfaces (APIs) with DB2® Information Integrator for Content Version 8 Release 4. The APIs and beans provide building blocks for creating applications that access content stored in heterogeneous content servers.

The following table summarizes and links to the high-level tasks for programming content management applications.

Programming tasks	What you can do	When to perform
"Working with a federated content server and federated searching (deprecated)" on page 9	Use a DKDatastoreFed object for a federated search. Run a federated search using the APIs by creating a federated query string first.	Perform when you run a federated search.
"Programming with the application programming interfaces (APIs)" on page 17	Use Java and C++ APIs to design your applications	Perform when you set up the Java and C++ environment, work with a DDO, XDO, and DKAny objects.
"Working with DB2 Content Manager Version 8.4" on page 63	Use the ICM connector APIs to build and deploy custom applications that access a DB2 Content Manager content server. Use the APIs to integrate your existing applications into a DB2 Content Manager content server.	Perform when you plan and create an application and work with the library server and the resource manager
"Searching for data" on page 171	Search for data by using parametric and text search	Perform when you search for data, use text and parametric search.
"Routing a document through a process" on page 209	Route documents through a business process by using document routing APIs	Perform when routing a document through worknodes, worklists, and processes.
"Understanding prefetching in DB2 Content Manager for z/OS" on page 237	Move objects that are currently located on slower media, like optical or tape, to faster media.	Perform when necessary.
"Working with other content servers" on page 243	Use the dkDatastore classes to define an appropriate content server for the content servers in your application.	Perform when working with OnDemand, ImagePlus® for OS/390®, and DB2 Content Manager for AS/400® content servers.
"Building DB2 Information Integrator for Content workflow applications (deprecated)" on page 271	Create or extend your own applications to use the DB2 Information Integrator for Content workflow support by using the DB2 Information Integrator for Content classes and APIs.	Perform when using DB2 Information Integrator for Content classes and APIs.
"Building applications with non-visual and visual JavaBeans" on page 291	Use the visual and non-visual JavaBeans™ provided in DB2 Information Integrator for Content to build Java-based or Web client applications.	Perform when using visual and non-visual beans to create applications.

Programming tasks	What you can do	When to perform
"Working with XML services (Java only)" on page 321	Import, store, update, retrieve, and export a wide variety of objects by representing objects as XML	Perform when working with XML services.
"Working with the Web services" on page 401	Integrate you applications or client process by using Web services	Perform when using Web services
"Working with the Java document viewer toolkit" on page 441	Use the document viewer toolkit to display, manipulate, and annotate documents.	Perform when using Java viewer.
"Working with the JSP tag library and controller servlet" on page 473	Use the Java™ Server Pages (JSP) tag library and a servlet to write JSP or servlets for Web applications.	Perform when necessary
"Troubleshooting content management applications" on page 487	Fix common problems that occur during programming.	Perform when necessary.

Important information for Linux users

This information is provided for your reference and might contain documentation about the following list of components that are NOT supported on Linux®:

- DB2 Information Integrator for Content advanced workflow
- All C++ connectors
- Remote Java connectors
- ImagePlus for OS/390 connector
- VideoCharger™ player

Deprecated and removed features

The deprecated and removed connector table lists the connectors, APIs, and services that were available in DB2 Content Manager Version 8.3 and their current status.

The following terminologies have been used:

active Active connector indicates that it continues to be maintained and, when appropriate, enhanced, in conjunction with current levels of the supported repository, development tools and operating environments.

deprecated

Deprecated connector is still available and maintained as part of the current release for supported environments, but in general is not recommended for use for new development; an alternative is usually indicated. While deprecated APIs are currently still maintained, they may be removed in future releases.

removed

Connectors that support systems that have reached end of support are removed in the current release.

DB2 Content Manager Version 8.4 does not support the DB2 connector, ODBC connector, JDBC connector, DD connector (Domino Doc), DL connector (Content Manager version 7), TS connector (Text Search), QBIC connector (Image Search), and watermarking from the DB2 Information Integrator for Content solution.

Additionally DB2 Content Manager Version 8.4 deprecates the federated connector (federation), federated advanced workflow, and the OD connector (DB2 Content Manager OnDemand).

The following table identifies the status and the changes to the DB2 Information Integrator for Content connectors in DB2 Content Manager Version 8.4.

Table 1. Deprecated and removed connectors

Connector	Description	Status
DB2	DB2 UDB	Removed
DD	Domino Doc	Removed
DL	IBM Content Manager V7	Removed
ICM	DB2 Content Manager V8	Active
IP	Image Plus (IP/390)	Active
JDBC	Relational connector	Removed
FED	Federation	Deprecated, use WebSphere II Content Edition
OD	CM OnDemand	Deprecated, use OnDemand Web Enablement Kit
ODBC	Relational connector	Removed
QBIC	Image search	Removed
TS	Text search	Removed
V4	CM iSeries V5 (VI/400)	Active

Advanced Workflow (MQ Workflow) is also deprecated in DB2 Content Manager Version 8.4.

All deprecated connectors and features are documented in the *Application Programming Reference* and does not necessarily report compiler deprecation warnings. The deprecated connectors and features are still supported at this time, they might be removed in a future release.

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. DB2 Information Integrator for Content supports the following content servers:

- DB2 Content Manager Version 8.4
- ImagePlus for OS/390
- OnDemand
- VisualInfo™ for AS/400

Applications that use DB2 Information Integrator for Content can create a federated content server, which acts as a common server. DB2 Information Integrator for Content federated classes enable federated searching, retrieval, and updating across several content servers.

The DB2 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. DB2 Information Integrator for Content keeps the schema and mappings, as well as other administration information in its administration database.

Dynamic data object concepts

In compliance with Object Management Groups' (OMG) CORBA Persistent Object Service and Object Query Service Specification, DB2 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 DB2 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)

A *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.

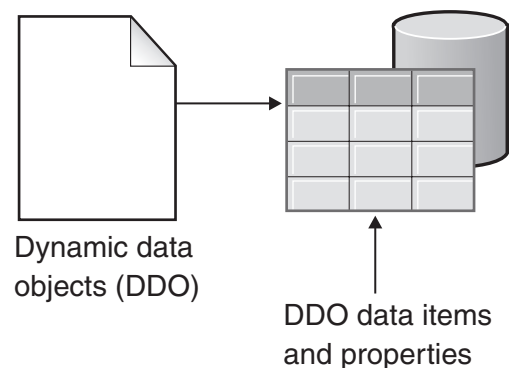


Figure 1. Dynamic data objects and items

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 shows an example of XDOs.

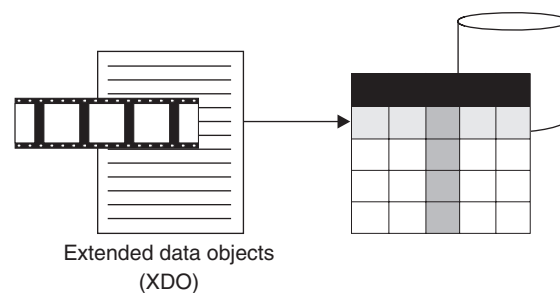


Figure 2. Extended data objects (XDOs)

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 Version 8, 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.

Content servers and DDOs

DDOs are created and dynamically associated with a content server.

The association between a DDO and a content server is established with the DDOs PID.

In general, an DB2 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

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.

A DDO corresponds to an item in DB2 Information Integrator for Content. 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 DB2 Information Integrator for Content.

Persistent identifiers (PID)

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

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

Working with a federated content server and federated searching (deprecated)

Federated searching is the process of searching for data in one or more content servers.

DB2 Information Integrator for Content is deprecated in DB2 Content Manager Version 8.4.

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 DB2 Information Integrator for Content framework.

Federated classes work across different content servers, such as ImagePlus for OS/390. 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.

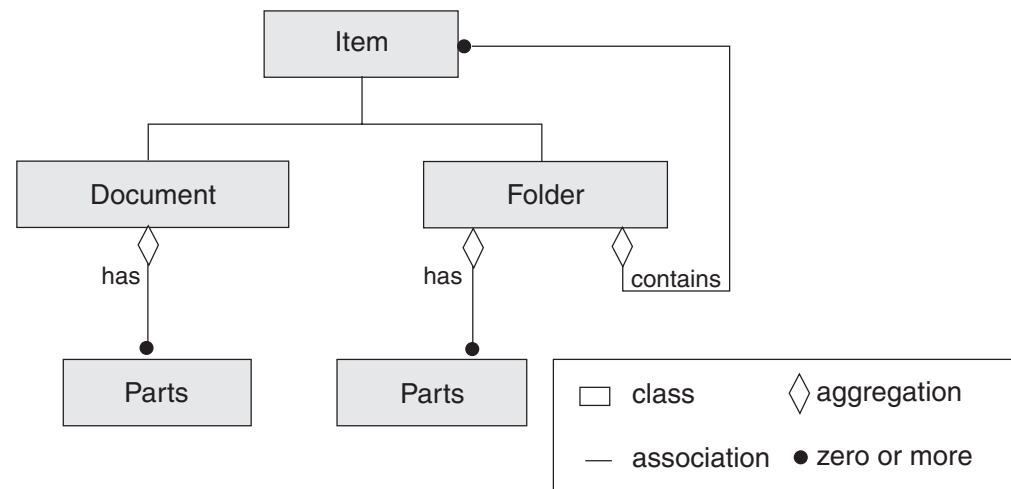


Figure 3. Federated document view

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 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 Version 8.4	library server	item type	attribute	item type view or item type subset
OnDemand	OnDemand server	application group folder	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
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

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 DB2 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

Federated content server schema mapping components include the user ID and content server registration information.

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 DB2 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 DB2 Information Integrator for Content.

The user ID and content server information is maintained in the DB2 Information Integrator for Content administration database.

Running federated queries

Running federated queries include creating a federated query string and then running the query.

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, DB2 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 4.

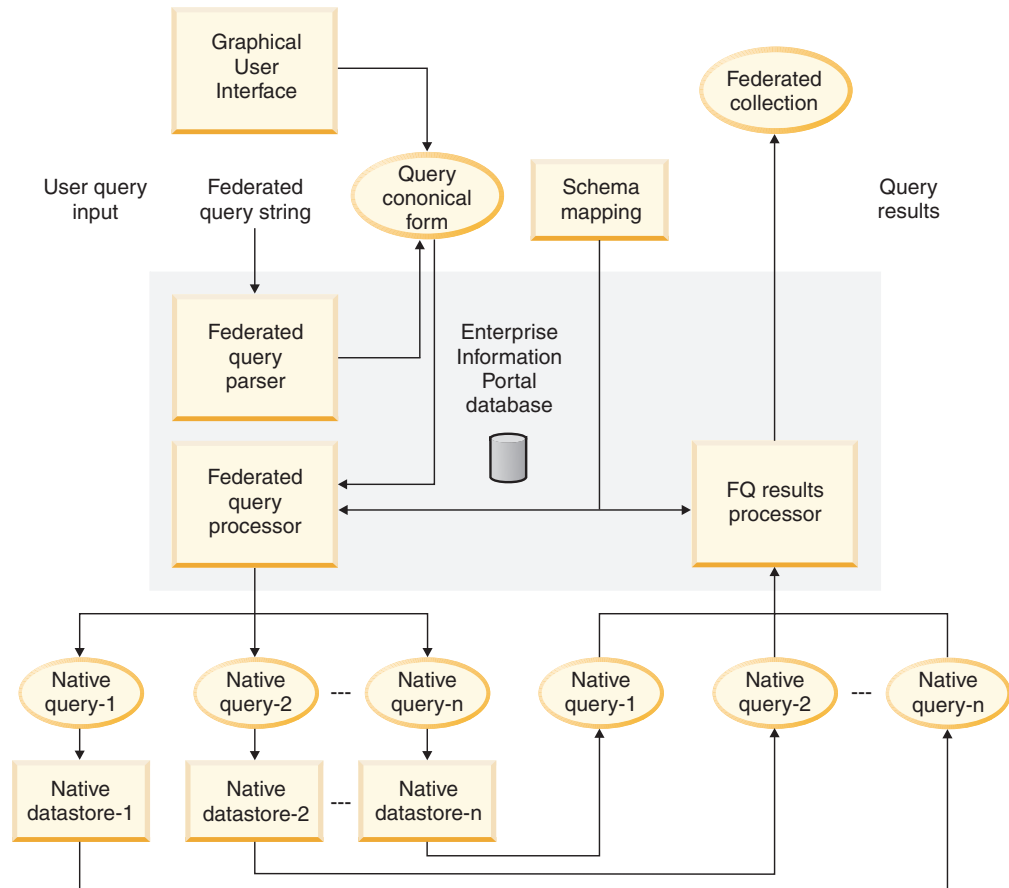


Figure 4. Federated query processing

Federated query syntax

Federated queries must have a 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=( [ASSOCIATED_ENTITY={associated_entity_name}]);]

```

```

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

```

The NOT operator is not supported in federated searches.

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 DB2 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 DB2 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 DB2 Content Manager Version 8 text-searchable item type or an Extended Search text-searchable entity.

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

Federated parametric and text query using the OR operator

```
"PARAMETRIC_SEARCH = ( ENTITY = FedTextResource,  
MAX_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)

DB2 Information Integrator for Content 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.

DB2 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 samples directory.

Working with system administration

DB2 Information Integrator for Content 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 DB2 Information Integrator for Content system administration client

You can customize the DB2 Information Integrator for Content system administration client to meet your requirements.

The DB2 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 DB2 Information Integrator for Content system administration client with your own dialogs.
- You can add new nodes to the hierarchy in the DB2 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 DB2 Information Integrator for Content system administration.

Programming with the application programming interfaces (APIs)

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

The APIs support:

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

Note that DB2 Content Manager uses Call Level Interface (CLI) to connect to the DB2 Universal Database™, so you should either use all DB2 CLI or all embedded SQL, not mixture, for connection management or transaction management within your applications.

Understanding differences between the Java and C++ APIs

There are some fundamental differences between 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++ APIs are not defined in the Java APIs. 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 APIs; 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 DB2 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 DB2 Information Integrator for Content APIs are contained in four packages.

The four packages that contain DB2 Information Integrator for Content APIs 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 package 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)

To setup the Java environment, you must import several packages and setup the Java environment variables.

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 DB2 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

To set up the environment for Solaris, use the shell script or set the environment variables.

In the Solaris 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 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 the planning and installing information and the administering information.

Important: The ICM connector does not support RMI configuration.

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

To set up the C++ environment, you must define the environment variables and the define the compiler flag.

When you set up your Windows or AIX environment, you must establish the settings described in this section. Table 3 on page 22 lists Library, AIX shared and DLL requirements.

Remember: To use C++, you must install DB2 Client support and the Client Configuration Assistant on all remote servers accessing the DB2 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 DB2 Information Integrator for Content database. For details, see the administering information.

Important: `cmbcm81x.lib` is for release build and `cmbcm81xd.lib` is for debug build, where *x* represents Microsoft® Visual C++ .NET Version 2003 (71) compiler or Microsoft Visual C++ Version 2005 (8).

Tip:

The Microsoft Visual Studio 2005 compiler can be installed on the same workstation as the Microsoft Visual Studio .NET 2003.

The Microsoft Visual C++ 2005 compiler can also be used to compile library server user exits for DB2 Content Manager installed on Windows.

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. For more information, check C++ sample makefiles.

To compile DB2 Content Manager applications using the Microsoft Visual C++ 2005 compiler, you must add the `MSVC80` and `MSVC71` compile flag in the `.mak` files or in the project files. For example, in the compile flag definitions, add `/D MSVC80` and `/D MSVC71`. If you do not define the compiler flag, your application will not compile. For more information, check C++ sample makefiles.

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.	cmbbcm81x.lib cmbbcm81xd.lib	cmbbcm81x.dll cmbbcm81xd.dll	libcmbbcm817.a
DB2 Content Manager Version 8	cmbbcm81x.lib cmbbcm81xd.lib	cmbbcm81x.dll cmbbcm81xd.dll cmbbcmfac81x.dll cmbbcmfac81xd.dll	libcmbbcm817.a libcmbbcmfac817.so
Federated	cmbfed81x.lib cmbfed81xd.lib	cmbfed81x.dll cmbfed81xd.dll cmbfedfac81x.dll cmbfedfac81xd.dll	libcmbfed817.a libcmbfedfac817.so
OnDemand	cmbod81x.lib cmbod81xd.lib	cmbod81x.dll cmbod816xd.dll cmbodfac81x.dll cmbodfac81xd.dll	libcmbod817.a libcmbodfac817.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

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

INCLUDE

INCLUDE=%INCLUDE%;%IBMCMROOT%

Setting the C++ environment variables for AIX

To set up the C++ environment variables for AIX, you can use the cmbenv81.sh batch file.

To set up the C++ environment variables:

Set the following environment variables:

1. In the AIX environment, you can use the cmbenv81.sh batch file to set up your development environment.
2. 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
```

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

Building C++ programs

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

There is no debug build on AIX.

Restriction: The Content Manager V8 C++ APIs do not support Unicode.

Setting the console subsystem for code page conversion on Windows

The example shows how to set up the console subsystem for code page conversion on Windows.

Example: C++

To set up the console subsystem for 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);
    ...
}
```

Java and C++ application performance

For DB2 Content Manager version 8.4, Java and C++ APIs provide a number of features that improve performance, usability, and scalability of API applications.

Java API global caching

You can use a global cache to improve the performance and scalability of Java API applications.

By default, every DKDatastoreICM instance stores certain objects (DKDatastoreICM instances) in memory within a local cache (meaning, other DKDatastoreICM instances cannot access it). These objects can include item type, item type view, component type, component type view, and so on. Caching improves response times by saving the Java API from repeatedly communicating with the library server to manifest these objects.

By enabling global cache, you can consolidate the caches of multiple DKDatastoreICM instances into one global area. Global caching prevents duplicating identical objects and maximizes performance and scalability of a

mid-tier environment. If you enable global cache, then the following types of objects are stored in the global area instead of the local ones:

- Attribute
- Attribute group
- Event type
- Link type
- XDO classification
- Mime type
- NLS keywords
- Semantic type
- Privilege
- Privilege group

To enable the global cache, edit *IBMCMROOT/cmgmt/connectors/cmbicmcache.ini* and set the following: `JAVAGlobalCacheEnabled=TRUE`.

You must clear the global cache whenever modifications (such as create, update, or delete) are made to any of the definition objects except for NLS keywords. For cache clearing examples, refer to the following examples.

Clearing the global and local caches of all datastores for an application that uses its own implementation of datastore pool

```
dsPool.clearCacheOfConnections(DK_CM_CLEAR_CACHE_ALL);
```

This example makes sure that the global cache is cleared once, and the local cache of all datastores are cleared.

Clearing both the local and global cache

```
dsICM.clearCache(DK_CM_CLEAR_CACHE_ALL);
```

Clearing only the local cache

```
dsICM.clearCache(DK_CM_CLEAR_CACHE_LOCAL);
```

Clearing only the global cache

```
dsICM.clearCache(DK_CM_CLEAR_CACHE_GLOBAL);
```

If the application does not set the option by using `setOption()` method but calls `clearCache()`, the `DK_CM_CLEAR_CACHE_ALL` method is used.

Native API renovated retrieve options

To improve usability, performance, documentation, and support of the native APIs, DB2 Content Manager renovates the retrieve options interface and underlying implementation. The renovated options give more granular control to the application for better performance control and enable a number of built-in optimizations.

Each retrieve option setting includes a detailed explanation, performance considerations, order of precedence considerations, data structures (how to find the data you requested in the DKDDOs), default value, behavior notes, detailed exception documentation, and more. Additionally, features are available, such as a simple, single general projection across all item types when building attribute filters when using the new tightly integrated projection lists.

DB2 Content Manager provides a DKRetrieveOptionsICM class and an integrated DKProjectionListICM class. The granular retrieve options are designed for selection of the required data based on performance-impacting incremental steps. In addition to built-in optimizations automatically enabled by using the options, you can attain the best possible retrieval performance by retrieving only the subset of data you plan to use and the performance considerations. Additionally, various combinations of retrieve selections are possible, such as retrieving one level of children and including child attributes and retrieving parts lists with parts attributes.

For details, see the detailed Javadoc for the DKRetrieveOptionsICM class.

Important: The bitwise integer retrieve options and no-options variants of retrieve methods that assume a bitwise 'int' option as default behavior are deprecated. No change to existing applications is required. The old bitwise integer retrieve options are still supported, and the behavior has not changed. However, for all new development or when modifying existing applications, use the renovated options for better performance.

C++ API scalability and performance: global caching and connection pooling

The global cache, datastore pool, and database connection pool improve the performance and scalability of C++ API applications.

To maximize performance and scalability, use the global cache, datastore pool, and database connection pools together for the following types of scenarios:

- Environments with multiple DKDatastoreICM instances, especially ones that use mid-tier servers such as DB2 Document Manager. These environments include custom applications such as Microsoft .NET, COM, and DCOM applications where the C++ API is running in a mid-tier server environment.
- Applications that use multiple threads, especially mid-tier process threads that repeat the following pattern:
 1. Connect to DB2 Content Manager.
 2. Perform an operation (such as run-time create, retrieve, update, delete, and query or administration/definition time operations).
 3. Disconnect from DB2 Content Manager.

The three C++ API features work as follows:

Global cache

By default, every DKDatastoreICM instance stores certain objects (DKDatastoreICM instances) in memory within a local cache (meaning, other DKDatastoreICM instances cannot access it). These objects can include item type, item type view, component type, component type view, and others. Caching improves response times by saving the C++ API from repeatedly communicating with the library server to manifest these objects.

By enabling global cache, you can consolidate the caches of multiple DKDatastoreICM instances into one global area. Global caching prevents the duplication of identical objects. If you enable global cache, then the following types of objects are stored in the global area instead of the local areas:

- Attribute

- Attribute group
- Event type
- Link type
- XDO classification
- MIME type
- NLS (National Language Support) keywords
- Semantic type

To enable the global cache, manually create a new configuration file, *IBMCMROOT/db2cmv8/cmgt/connectors/cmbicmcache.ini*, and include the following line in the .ini file : `CPPGlobalCacheEnabled=TRUE`.

You must clear the global cache whenever modifications (such as create, update, or delete) are made to any of the definition objects except for national language support keywords.

Datastore pool

Within a process, threads that need to access the DB2 Content Manager server can obtain connected and logged-on DKDatastore objects from a datastore pool, then return them to the pool when they are done. Particularly, datastore pools can reduce response times on the mid-tier server. In combination with the global cache, the datastore pool reduces the number of connections and calls to the library server.

A datastore pool requires no configuration.

Database connection pool

The database connection pool is used internally by the DB2 Content Manager Version 8 API to pool database connections to the library server database. This is similar to how the datastore pool is used by the application to pool datastores. The pool holds active connections to the library server database that are used when needed. The pool makes accessing the library server faster because that database connection is not required. It also minimizes the number of database connections.

To enable the database connection pool, modify the configuration file: *IBMCMROOT/db2cmv8/cmgt/connectors/cmbpool.ini*, by inserting the following line in the .ini file:

```
CPPDBConnectionPoolMaxSize = N
```

where *N* represents the largest number of database connections that the administrator wants to allow for any given C++ application process. Setting the `CPPDBConnectionPoolMaxSize` property to 0 disables the database connection pool.

Requirement: The `CPPDBConnectionPoolMaxSize` property should not exceed 70 if the log level in the `cmblogconfig.properties` is set to be any one of the following values: INFO, TRACE_NATIVE_API, TRACE_ENTRY_EXIT, TRACE or DEBUG.

Recommendation: If the database connection pool is enabled, make sure that the application closes the pool of connections when it is terminating. The enablement ensures a clean shut down with a timely release of resources. The following

static method in the DKDatastoreICM instance should be used to close all the connections in the database connection pool:

```
static DKEXPORT void closeDBConnectionPool();
```

Clearing both the local and global cache

To clear local and global cache:

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();  
dsICM->setOption(DK_CM_OPT_CLEAR_CACHE, DK_CM_CLEAR_CACHE_ALL);  
dsDefICM->clearCache();
```

Clearing only the local cache

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();  
dsICM->setOption(DK_CM_OPT_CLEAR_CACHE, DK_CM_CLEAR_CACHE_LOCAL);  
dsDefICM->clearCache();
```

Clearing only the global cache

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();  
dsICM->setOption(DK_CM_OPT_CLEAR_CACHE, DK_CM_CLEAR_CACHE_GLOBAL);  
dsDefICM->clearCache();
```

If the application does not set the option by using the setOption() method but calls clearCache(), the DK_CM_CLEAR_CACHE_ALL method is used.

Clearing the global and local caches of all datastores for an application that uses its own implementation of datastore pool

For the first DKDatastoreICM instance in the pool, add this code to your application:

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();  
dsICM->setOption(DK_CM_OPT_CLEAR_CACHE, DK_CM_CLEAR_CACHE_ALL);  
dsDefICM->clearCache();
```

For the remaining DKDatastoreICM instances:

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();  
dsICM->setOption(DK_CM_OPT_CLEAR_CACHE, DK_CM_CLEAR_CACHE_LOCAL);  
dsDefICM->clearCache();
```

This example makes sure that the global cache is cleared once, and the local cache of all datastores are cleared.

Clearing the global and local caches of all datastores for an application that uses a C++ API implementation of datastore pool

To clear global and local cache:

```
dsPool->clearCacheOfConnections();
```

Timing out connections with C++ API

DB2 Content Manager enables timing out connections in the database pool.

The C++ API supported a mechanism to pool database connections from DB2 Content Manager Enterprise Edition Version 8 Release 3 Fix Pack 2 and later. The database connection pool is used internally by the DB2 Content Manager Version 8 C++ API to pool database connections to the library server database. The connection pool holds the active connections to the library server database that are

used when required. It fetches a connection from the pool when a transaction (implicit or explicit) is started and returns the connection to the pool when the transaction is complete. This action enables maximum reuse of the connections in the pool and minimizes the number of database connections required.

To enable the database connection pool, modify the configuration file located at `%ibmcmroot%/db2cmv8/cmgmt/connectors/cmbpool.ini`, and enter the following line:

```
CPPDBConnectionPoolMaxSize = N
```

where *N* represents the largest number of database connections that the administrator wants to allow for any given C++ application process. Setting `CPPDBConnectionPoolMaxSize` to 0 disables the database connection pool. See the "C++ API scalability and performance improvements:global caching and connection pooling" section in *DB2 Content Manager Enterprise Edition Version 8 Release 3 Fix Pack 2 Readme* for more information on the type of scenarios under which the C++ API Database Connection pool must be enabled.

DB2 Content Manager Enterprise Edition now enables a timeout mechanism to remove the unused connections in the connection pool for C++ API. Timing out a connection results in the connection being disconnected and removed from the pool. The timeout value, in minutes, can be set in the `cmbpool.ini` file (located at `%ibmcmroot%/cmgmt/connectors`) and is effective only if the C++ database connection pool is enabled. A new property called `CPPDBConnectionPoolTimeOut` is available to time out unused connection in the database connection pool.

Important: The values for both `CPPDBConnectionPoolMaxSize` and `CPPDBConnectionPoolTimeOut` are set in `cmbpool.ini` file.

Unused connections are connections that are not fetched (or used) from the pool for the duration indicated by the `CPPDBConnectionPoolTimeOut` property. Each connection has a last used time, which is used to determine if the connection was not used for the duration indicated by the `CPPDBConnectionPoolTimeOut` value. If the connection has not been used for the duration indicated by `CPPDBConnectionPoolTimeOut` property, the connection is removed from the pool. Removing unused connections also de-allocates the resources in both the library server and API. The database timeout mechanism is disabled by setting the value for `CPPDBConnectionPoolTimeOut` property to 0. If this value is not set in the `cmbpool.ini` file and database connection pool is enabled, a default value of 5 minutes is used.

Important: You must manually update the `cmbpool.ini` file to set the appropriate timeout values for your environment.

Indicating elements fetched when retrieving the list of ACLs

DB2 Content Manager provides APIs for the `DKAuthorizationMgmtICM` class that enables applications to indicate the elements that need to be fetched while retrieving the list of ACLs in the system.

The APIs, `listAccessControlLists(*, retrievalOption)` and `listUserAccessControlLists(*, retrievalOption)` provide more flexibility to applications in determining what needs to be fetched when they list ACLs (* means 0 or more parameters for `listAccessControlLists` and `listUserAccessControlLists`).

DB2 Content Manager also provides another API for the DKAutorizationMgmtICM class, listPrivilegeSets(*, options) that enables applications to have more flexibility in determining what needs to be fetched when they list privilege sets. Options can be one of the following:

DK_ICM_PRIVILEGE_SET_RETRIEVAL_OPTION_NAME_ONLY

Returns a collection of privilege set objects (instance of DKPrivilegeSetICM). Each privilege set contains the privilege set name only.

DK_ICM_PRIVILEGESET_RETRIEVAL_OPTION_ATTRIBUTES_ONLY

Return a collection of privilege set objects (instance of DKPrivilegeSetICM). Each privilege set contains the following properties:

- Privilege set name
- Privilege set description
- Collection of domains to which the privilege set belongs

DK_ICM_PRIVILEGESET_RETRIEVAL_OPTION_ALL

Returns a collection of privilege set objects (instance of DKPrivilegeSetICM). Each privilege set contains the following:

- Privilege set name
- Privilege set description
- Collection of domains to which the privilege set belongs
- Collection of privileges that form the privilege set (each privilege is an instance of DKPrivilegeICM class)

Important: The DKPrivilegeSetICM objects returned by DK_ICM_PRIVILEGE_SET_RETRIEVAL_OPTION_NAME_ONLY and DK_ICM_PRIVILEGESET_RETRIEVAL_OPTION_ATTRIBUTES_ONLY are incomplete because all its properties are not filled. Do not use the incomplete object to perform any updates. Instead, use the retrievePrivilegeSet(long id) API to retrieve the fully formed object, update the fully formed object, and use this object to update the privilege set information in the persistent store.

Both the APIs, listAccessControlLists(*, retrievalOption) and listPrivilegeSets(*, options) are available for both Java and C++. For more information, see the *Application Programming Reference*

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 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.

Handling exceptions

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

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.

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 DB2 Information Integrator for Content and CM:

Example: Java

```
try{
    ... EIP API Operations ...
}
catch (DKException exc){
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    System.out.println("");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    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("XXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("    Name: " + exc.getClass().getName());
    System.out.println(" Message: " + exc.getMessage());
    exc.printStackTrace();
    System.out.println("-----");
}
```

Example: C++

```
try{
    ... EIP API Operations ...
}
catch (DKException &exc){
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    cout << endl;
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    cout << "X    !!! Exception !!!    X" << endl;
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXXX" << 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).

The constants specified are in the form of DK_CM_ (Common constants) or DK_XX_ (where the XX indicates different content servers). See Table 4 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 DKConstantX.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.

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 4 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 4. Server type and class name terminology

Content server	Class name
DB2 Content Manager Version 8	DKDatastoreICM
OnDemand	DKDatastoreOD
DB2 Content Manager for AS/400 (VisualInfo for AS/400)	DKDatastoreV4

Table 4. Server type and class name terminology (continued)

Content server	Class name
ImagePlus for OS/390	DKDatastoreIP

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 ICMNLSDB, the user ID ICMADMIN and password PASSWORD. The example creates a DKDatastoreICM object for the CM content server, connects to it, works with it, then disconnects from it.

Example: Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //Create datastore object
dsICM.connect("ICMNLSDDB","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("ICMNLSDDB","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 as you used to establish a connection 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.

Important: A valid `DKDatastoreICM` object is already created in a variable called `dsICM`.

Example: Java

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

Example: 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).

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.

Example: 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()+"'");
}
```

Example: 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.

Example: Java

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.

```

. . .
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 . . .
    }
    // ----- 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.

Example: C++

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

```

// 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);

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

```

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.

Working with dynamic data objects (DDOs)

You use the DKDDO class for dynamic data objects (DDOs) in your IBM DB2 Information Integrator for Content applications.

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.

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.

Example: Java

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

Example: 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):

For a more efficient constructor, pass in the exact number of attributes you want (for example, ten):

Example: Java

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

Example: C++

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

Important: DB2 Content Manager V8 users (DKDatastoreICM API) as well as the DKDatastoreOD API users should always create DDOs using the DKDatastoreXX::createDDO() methods rather than using DKDDO constructors directly. The createDDO() method creates the appropriate subclass, initializes meta-data structures, populates defaults, and so on.

The following example creates a DKDDO by passing in both content server and object type for Content Manager Version 8:

Example: Java

```
// Given a connected DKDatastoreICM object named "dsICM"  
// Create a DDO representing an instance of an item type  
DKDDO ddo=dsICM.createDDO("ICMSAMPLE",DKConstant.DK_CM_DOCUMENT);
```

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

Example: C++

```
// Given a connected DKDatastoreICM object named "dsICM"  
// Create a DDO representing an instance of an item type  
DKDDO* ddo = dsICM->createDDO("ICMSAMPLE", DK_CM_DOCUMENT);
```

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

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*)

Creating a persistent identifier (PID)

Each DDO must have a persistent identifier (PID). The PID contains information about a particular content server, like the datastore name, datastore type, and the object type .

The PID identifies the DDO's persistent data location. For example, in earlier DB2 Content Manager content server, the PID contained the item ID. The item ID is one of the most important parameters for the retrieve, update, and delete functions in earlier DB2 Content Manager content server. In DB2 Content Manager V8, the PID has other information needed such as item ID and component ID.

The DB2 Content Manager V8 datastore createDDO() method will create a PID object for the DDO object created. During the add operation in DB2 Content Manager V8, the datastore will populate the PID object with information such as the item ID.

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

Example: 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();
```

Example: 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();
```

DB2 Content Manager V8 connector has a PID class called DKPidICM which is a subclass of DKPid. The DKPidICM object is used by the DKDDO as well as subclasses of dkResource .

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.

Important: Assumes that you have a connected DKDatstoreICM in variable *dsICM*. The user-defined item type *S_withChild* must be defined with varchar *S_varchar*, long integer *S_integer*, short integer *S_short*, and time *S_time*, as defined in the *SItemTypeCreationICM* API education sample.

Example: 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.

Example: 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.

Example: 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.

Example: 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.

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

Example: 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.

Example: 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.

Example: 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.

Example: 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.

Example: 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.

Example: 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 (`delete myddo`). 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 destructor deletes object, all internal metadata structures, content in memory, and all DDO data items and structures owned by the DDO.

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. Depending on the type of resource item, the XDO can be extended further.

DB2 Content Manager V8: DB2 Content Manager V8 requires XDOs to be of the correct subclass of `DKLobICM` based on the item type's XDO classification and associated subclass. You must always create DKDDOs using the datastore object's `createDDO()` methods (`DKDatastoreICM::createDDO()`). This allows the system to automatically create an instance of the appropriate sub-class of DKDDO or `DKLobICM`, initialize other important information, and properly setup greater functionality such as resources, document model, and folders. DKDDOs from resource or document part item type classifications (returned from `DKDatastoreICM.createDDO()` methods) can be cast to the actual subclass depending on the item type classification and XDO class. For more information on creating items in general and the `DKDatastoreICM.createDDO()` function, see the `SItemCreationICM` and `SResourceItemCreationICM` API samples.

Example: Java

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

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 DB2 Content Manager V8 Connector, use `DKPidICM` to represent the PID of a `dkResource` object (which is an XDO typically instantiated as a `DKLobICM` or subclass).

Remember: Use the `DKDatastoreICM::createDDO()` methods to construct your resource-enabled DDOs which also creates the PID for you.

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:

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.

Example: Java

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


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.

Important: The user-defined item type, `S_docModel`, must be defined in the system, classified as Document Model, and support `ICMBASE` and `ICMBASETEXT` part types, as defined in the `SItemTypeCreationICM` API education sample

Example: 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.

Example: 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 Version 8. For examples for earlier Content Manager and other content servers, see *Representing items using DDO*, working with other content servers, 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.

Important: 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.

Example: 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.

Example: 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");
("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.

When storing objects, you can specify an alternate resource manager collection in your custom application. For more information, see DKLobICM in the *Application Programming Reference*.

Important: 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.

Example: 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.

Example: 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.

Important: 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 *SItemCreationICM* 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*.

Example: Java

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

// Create annotation part
DKLobICM annot = (DKLobICM) dsICM.createDDO("ICMANNOATATION",
                                             DKConstant.ICM_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.

Example: C++

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

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

// Set annotation 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 persistent 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

The following example retrieves, updates, and deletes an XDO in Content Manager.

Important: The user-defined item type S_text of classification resource must be defined in the system, as defined by the SItemTypeCreationICM API education sample. Additionally, the resource manager and SMS collection definitions must be 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.

Example: Java

```

// Given: String pidString

// Re-create Blank DDOs for Existing Item (SItemRetrievalICM sample)
DKLobICM lob = (DKLobICM) dsICM.createDDOFromPID(pidString);

// Retrieve the item with the resource content
DKRetrieveOptionsICM retrOpts = DKRetrieveOptionsICM::createInstance(dsICM);
retrOpts.baseAttributes(true);
retrOpts.resourceContent(true); // Load into memory
lob.retrieve(retrOpts.dkNVPair());

// 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.

Example: C++

```

// Given: DKString pidString

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

```

```

// Retrieve the item with the resource content
DKRetrieveOptionsICM* retrOpts = DKRetrieveOptionsICM::createInstance(dsICM);
retrOpts->baseAttributes(TRUE);
retrOpts->resourceContent(TRUE); // Load into memory
lob->retrieve(retrOpts->dkNVPair(), retrOpts->dkNVPairLength());

// 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);
delete(retrOpts);

```

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

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.

Important: 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.

Example: 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.

Example: C++

```

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

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

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

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

// Access the DKParts attribute
DKParts* dkParts = (DKParts*)(dkCollection*)
    ddoDocument->getData( ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS));
// Add parts to document
dkParts->addElement((dkDataObjectBase*)(DKDDO*)base);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText1);

```

```

dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText2);

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

```

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

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.

Example: 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.

Example: 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.

Important: 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.

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

Example: 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.

Example: 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.

Example: 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 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.

Example: 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, 1970 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.

Example: 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.

Example: 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.

Example: 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.

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:

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

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

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

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

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

Displaying DKAny

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

Example: 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:

Example: 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

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

Example: 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.

Important: For 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).

Example: Java

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

Example: 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.

Example: 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
....
```

Example: 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.

Example: 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
```

Example: 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.

Example: C++

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

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

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

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

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

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


```

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:

```

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.

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 5 on page 61 shows the structure and behavior of `DKFederatedCollection`.

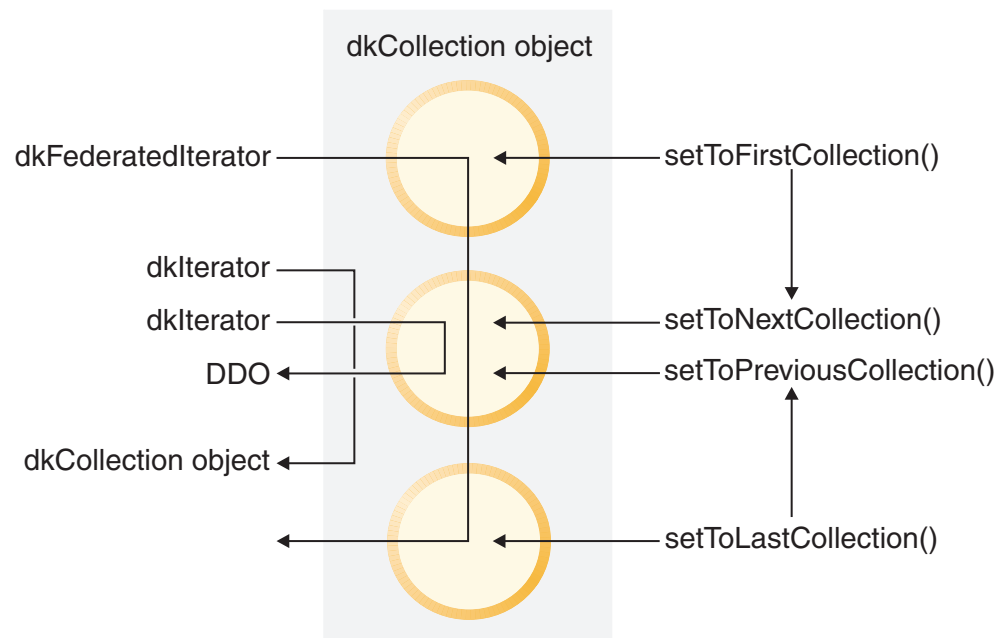


Figure 5. Structure and behavior of DKFederatedCollection

In Figure 5, 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.

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.

`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.

Differences between `dkResultSetCursor` and `DKResults`

There are some fundamental differences between `dkResultSetCursor` and a `DKResults` collection.

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 contains are fetched one at a time. It can also be used to run a query again to get the latest results.
- 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: For query information about DB2 Content Manager V8, see “Querying a content server” on page 61.

Working with DB2 Content Manager Version 8.4

This section describes the DB2 Content Manager Version 8 Release 4 connector (ICM connector) application programming interfaces (APIs). The ICM connector is an extension of the DB2 Information Integrator for Content framework, so it is essential that you understand the DB2 Information Integrator for Content framework.

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.

Important: The ICM connector does not support RMI configuration.

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 DB2 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 6 on page 64 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.

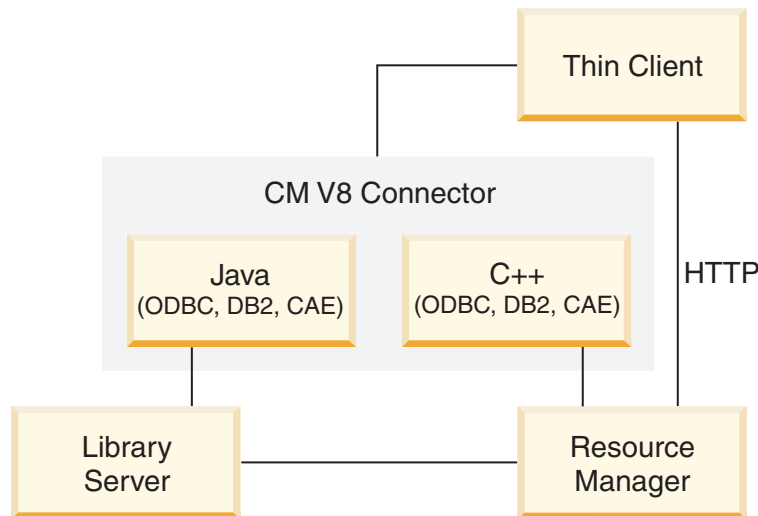


Figure 6. System configuration

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.

Items

An *item* is the basic entity managed by the library server.

The 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 item ID is stored by DB2 Content Manager and used to locate the item within the library server. When writing your application, you use the item ID 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 7 on page 66 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.

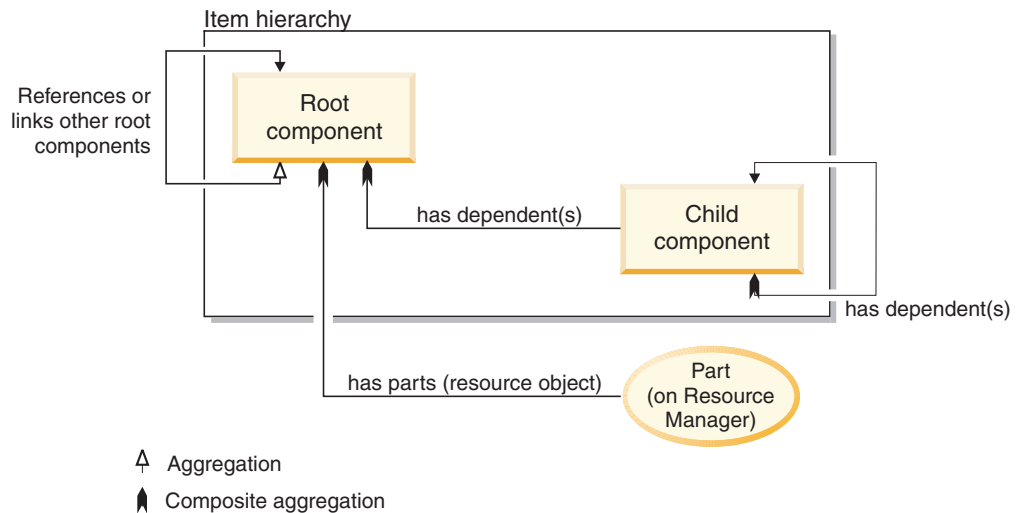


Figure 7. The DB2 Content Manager meta-model: A logical view.

Figure 8 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.

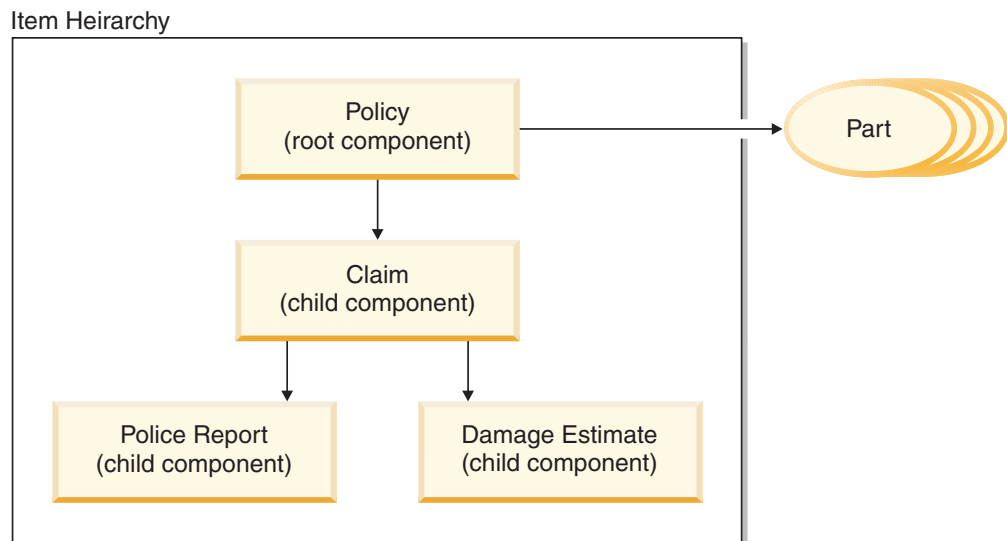


Figure 8. Components hierarchy

For more information about DB2 Content Manager data modeling concepts, see 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

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.

You can use a link to model one to many associations between items. As shown in Figure 7 on page 66, 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-contains 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 7 on page 66. 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.

Enabling auto-linking with version-enabled item types

DB2 Content Manager provides auto-linking with version-enabled item types.

The earlier versions of DB2 Content Manager provides a more restricted implementation of auto-linking, which was called auto-foldering. Auto-foldering was restricted to linking folders only. However, DB2 Content Manager enables auto-linking with version-enabled item types since Version 3 Fix Pack 4.

The auto-linking feature enables any two item types to be linked by one or more attributes. The source item type of the link is the container (such as the folder) and the target is the containee (such as the document item type whose items are contained in the folder items). If two items are linked, all the linking attributes must match. Multiple levels of auto-linking are possible, such as document type

Doc A can be automatically linked to Folder 1, and Folder 1 is linked to Folder 2, which is automatically linked to Folder 3, and so on.

Auto-linking can be enabled at the root or child component level, or both. A document item type can also be linked to a folder item type by attributes in the root or first-level child component in the heirarchical structure.

Since Fix Pack 4, DB2 Content Manager enables linking of version-enabled target item types. The source (or folder) item type to be linked to must not be enabled for versioning. The links for the target (or Document) item type are maintained by using the attributes of the current version of the item.

During automatic linking of a version-enabled target item type, DB2 Content Manager applies the following rules:

Update to the latest version of target item

If the update is for a linking attribute, DB2 Content Manager:

- Removes all links to current previous versions
- Creates the new version with the update
- Creates the links again by using the latest version of the target item to any of its source types

Deletion of the latest version of a document

To delete the latest version of a document, DB2 Content Manager:

- Removes all links to the items
- Deletes the latest version
- Restores the link between its previous version and any of its source item types

Re-index of a document

To re-index a document, DB2 Content Manager:

- Moves only the latest version to the new item type. The links are broken between the old item type and any source item types for that item.
- Establishes new links between the new item type and any source item types that it might have

Restrictions:

When you define an auto-linking rule that involves a child component attribute linked to a source root attribute, the minimum cardinality for the child component must be set to 1. The maximum cardinality can be set to 1 or greater. However, you must maintain the uniqueness of the attribute of source item types if that attribute is unique.

There are some restrictions for linking attributes. The linking attributes type cannot be time stamped. When you define an auto-folder link between two item types, you can have a root-to-root and child-to-root link concurrently, but the linking attribute cannot be the same for both the root and the child. To enable auto-linking at the child component level, the linking attribute must be set at the root level also.

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:

- 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.

- An item type can be configured to keep only a limited number of versions for each item. If an update to an item exceeds the maximum number of allowed, the oldest saved version is dropped and a new version is created by the system.
- If a version-enabled item is re-indexed, all previous versions of the item are automatically deleted.
- Child components of an item inherit the version of their parent component.
- The version of a child component type cannot be changed, since it follows the versioning of its parent type.
- Part-level versioning rules can be obtained from the item type relation object that represents the types.

Below is a sample that demonstrates how to work with version numbers.

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

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, and 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 9 on page 71 shows an example of how the system determines user's access rights to an item based on privileges and ACLs.

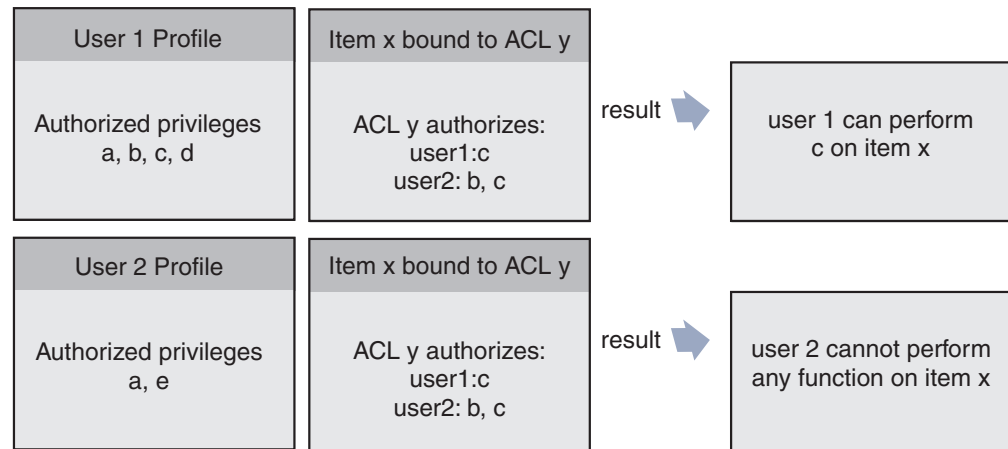


Figure 9. Access control diagram

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.

In the Content Manager ACL model, three privileges are always checked within the ACL code of the item type, even when item level ACL is used. These three privileges are: ItemTypeQuery, ItemAdd, and ItemMove.

The ItemTypeQuery privilege for an item type (or view of an item type) determines whether you can retrieve the item type or view. If you do not have this privilege in the item type ACL, you cannot see the item type when you list item types. This implies that all items in this item type could not be accessed. Therefore getting any item from the item type through retrieve or query is impossible, no matter what the ACL on the item is.

The ItemAdd and ItemMove privileges are always checked in the item type's ACL, even when using item level ACL. You must have the ItemAdd privilege within the item type ACL to be allowed to create items within the item type. Similarly you must have the ItemMove privilege in the target item type ACL to be able to reindex an item into this item type.

A group of privileges assigned to a user is a privilege set. A user's general privilege set determines the maximum amount of operations that user can execute. 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 Administration privileges

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: None

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 5. Privilege Codes

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

Users and user groups

A user group is solely a convenience grouping of individual users who perform similar tasks.

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 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.

Adding and removing users in large user groups using the C++ APIs

When you add a user to a large user group with about 1000 users, it takes around 45 minutes to complete the update. This delay happens in C++ only when the following methods are used:

```
userGroup->addUser(userDef);  
userMgmt->update(userGroup);
```

To improve the time to complete the update, create the following two new APIs:

```
void DKUserMgmtICM::update(dkUserGroupDef* pUserGroup, dkCollection* pUserColl,  
                           short sAction)  
void DKUserMgmtICM::update(dkUserGroupDef* pUserGroup, dkCollection* pUserColl,  
                           short sAction, short sOption)
```

Using the above APIs can significantly reduce the time it takes to complete the task.

Access control lists (ACLs) and ACL binding level rules

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.

Access control lists (ACLs)

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 is used at run time to determine what CRUD operations a user can execute on a particular item or item type. 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 user ACL is any ACL created by an end user with UserACLOwner privilege and can be assigned only to items. Users can search on user ACLs. User ACLs do not display in the system administration client. A user who is listed in the user ACL and who has UserACLOwner privilege, or an administrator, can modify a user ACL by using the APIs.

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.

An ACL contains one or more ACL rules. 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.

ACL binding level rules

ACL Binding Level is a configuration parameter that allows the user to choose that ACL that is used for ACL check. An ACL check is an additional check used at run time to determine what CRUD operations a user can execute on a particular item or item type. In addition to satisfying a general privilege check, the ACL check must be satisfied for the particular item or item type, depending on the action.

ACL checking is based on the following rules:

- The privileges that the users get for ACL X are the intersection between the privileges in the user general privilege set and the privilege set in the ACL.
- If a user participates in an ACL as both an individual user and as part of a group, the user ACL rule will override the user group ACL rule.
- When **PubAccessEnabled** parameter is enabled, and an ACL X contains an ACL rule for ICMPUBLIC, the set of privileges the user gets in ACL X are the UNION of:
 - the privileges given by user or user group ACL rules
 - the privileges given by the ACL rule to ICMPUBLIC
- Users with ItemSuperAccess privilege can bypass ACL checking. The user will be automatically added to every ACL, with all privileges in the general privilege set from his user profile.

The configuration parameter, **ACL Binding Level** allows the user to choose the ACL that is used to do the privilege check on CRUD operations. The binding level options are:

Item type level

The ACL is taken from the item type view definition. If no view is specified, or if the item type is a part, then the ACL is taken from the base item type view (which is the same as the item type ACL).

Item level

The ACL is taken from the item ACL. The item's ACL either is set by the application during item creation, or is defaulted to the item type ACL or the user's default ACL, depending on the item type definition.

Mixed (Item or item type)

The ACL is taken either from the item type view ACL or the item ACL, depending on the item level ACL flag set during item type definition. Mixed ACL binding level is the default setting for Content Manager Version 8.

Library level

The ACL is taken from the library ACL code from the library server configuration parameter.

The following “rules” describe how the ACL privilege check is performed based on the ACL binding level and item type classification.

Item rules

Item type level

Check privileges using the ACL from the provided item type view, or base view if none provided (ACLCODE from ICMSTITVIEWDEFS table).

Item level

Check privilege using the ACL from the item (ACLCODE from ICMSTITEMS001001 table).

Mixed level

Check the item level ACL flag from the item type definition (ITEMLEVELACL from ICMSTITEMTYPEDEFS table) to determine whether ACL binding is done at the item type level or the item level. For system predefined document routing itemtypes (ROUTINGPROCESS, WORKNODE, WORKLIST), the item level ACL flag is true (ACL is checked at item level).

Library level

Check the privilege using the library ACL code from the library server configuration (LIBRARYACLCODE from ICMSTSYSCONTROL table).

Part rules

Item type level

Check privilege using the ACL from the item type relation for the part (DFLTACLCODE from ICMSTITEMTYPEREL table).

Item level

Check privilege using the ACL from the item (ACLCODE from ICMSTITEMS001001 table).

Mixed level

Check the item level ACL flag from the part item type definition (ITEMLEVELACL from ICMSTITEMTYPEDEFS table) to determine whether ACL binding is done at the item type level or the item level. For system predefined part types (ICMBASE, ICMBASETEXT, ICMBASESTREAM, ICMNOTELOG, ICMANNOTATION), the item level ACL flag is false (ACL is checked at item type level).

Library level

Check the privilege using the library ACL code from the library server configuration (LIBRARYACLCODE from ICMSTSYSCONTROL table).

ACL configuration for document model

At run time, the library server always checks the user privileges with the ACL.

Each item and item type view has an ACL associated with it. The ACL that must be used for the access control check depends on the ACL binding. At run time, the library server:

- Checks the ACL at item type level when you are creating a document or retrieving item type information. If you access an item type view, the library server checks the ACL of the item type view. However, to retrieve system-defined parts, item type information does not require any ACL or privilege checking. For user-defined part types, the library server must check part type ACL.
- Retrieves the ACL based on the ACL binding to check the ACL and the privileges when you are searching and retrieving a document, updating a document with parts, and for all other runtime functions. You can set the ACL binding and the default ACL in the ACL window of an item type definition from system administration client.

The ACL checking for the document and the parts are separate and independent. Therefore, a user might be able to update either the document or the resource parts based on the ACL or the privilege.

Each view is assigned an ACL for an item type. The ACL checking for document parts is independent of the document views. There is no view for document parts. However, it is possible to create views on user-defined part types.

The item type level ACL that the library server uses for the document is set at the Access Control window. For the base view and for other item type views, the ACL is set in the item type subset window of the system administration client.

The ACL is specified in the item type relation for each part type when the document item type is created. The item type level ACL used by library server for the resource parts are from the Document Management window in the system administration client

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.

Determining the features of your application

The features of your application depends 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.

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 6 contains `DKException` information.

Table 6. DKException information

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

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.

Enabling IPv6 dual-stack support

DB2 Content Manager Version 8.4 provides Internet Protocol Version 6 (IPv6) dual-stack support in addition to Internet Protocol Version 4 (IPv4).

IPv6 is the next generation protocol created by Internet Engineering Task Force (IETF) to replace the current protocol, IPv4. IPv6 eliminates the shortage of IPv4 addresses that are required by all new machines connected to the Internet. IPv6 also enhances the routing and network auto-configuration of IPv4.

IPv4 has 32 bit addresses. IPv6 provides an almost unlimited number of addresses. IPv6 addresses are 128 bits. In addition to solving the network address shortage problem, IPv6 also provides:

- Automatic configuration
- Site renumbering
- End-to-end IP security
- Mobility with route optimization

Important: DB2 Content Manager Version 8.4 provides IPv6 dual-stack support for ICM connectors only for both the Java and C++ APIs. For example, Fed connectors are not supported for IPv6.

Important: The guidelines for using IPv6 addresses with URL strings are:

- In an URL string, an IPv6 address is enclosed in brackets. If you are using a hostname, no brackets are required even though the host has an IPv6 address. For example, the URL using IPv4 address 9.67.122.66 is `http://9.67.122.66:8080/index.html`. If you are using a host name like `myhost.company.com`, the URL is `http://myhost.company.com:8080/index.html`. However, the URL using IPv6 address `2001:0DB8:4545:2::09FF:FEF7:62DC` is `http://[2001:0DB8:4545:2::09FF:FEF7:62DC]:8080/index.html`
- If you are using IPv6 address in DB2 JDBC URL string, the IPv6 address must be enclosed in square brackets. For example, the connect string must be in the format:
`JDBCURL=jdbc:db2://[2007::9:181:141:150]:50000/icm1sdb`

To enable IPv6 support, Java (1.4 or later) provides two system properties:

java.net.preferIPv4Stack

If an application has a preference to use IPv4 sockets only, set this property to true. If this property is set to true, the application does not communicate with IPv6 hosts.

java.net.preferIPv6Addresses

Set this property to change the preference to use IPv6 addresses over IPv4 addresses.

While you are running a Java application by using an ICM connector, set the `java.net.preferIPv4Stack` and `java.net.preferIPv6Addresses` properties so that the Java application communicates with either IPv4 or IPv6 or both IPv4 and IPv6 dual-stack hosts.

The following table summarizes the results of various combinations of `java.net.preferIPv4Stack` and `java.net.preferIPv6Addresses` properties when you are running a Java class on a dual-stack node. Example: `java -Djava.net.preferIPv4Stack=false -Djava.net.preferIPv6Addresses=true <class name>`

Table 7. Preferences to use IPv4 or IPv6

<code>java.net.preferIPv4Stack</code>	<code>java.net.preferIPv6Addresses</code>	default address
true	true	IPv4
true	false	IPv4
false	false	IPv4
false	true	IPv6

IBM DB2 driver for JDBC and SQLJ support in DB2 Content Manager Version 8.4

DB2 Content Manager Version 8.4 supports the IBM® DB2® Driver for JDBC and SQLJ (type 2 and type 4). This driver is sometimes referred to as the JCC JDBC driver. This driver change impacts the CM 8 connector (ICM) and the federated connector.

The DB2 database deprecated the JDBC driver that DB2 Content Manager required for Version 8.3. DB2 Content Manager Version 8.4 no longer supports the DB2 JDBC Type 2 Driver (`com.ibm.db2.jdbc.app.DB2Driver`).

In Version 8.4, DB2 Content Manager supports only the IBM DB2 Driver for JDBC and SQLJ (type 2 and type 4).

The type 4 driver is a pure Java driver, which means that the application requires only the JAR files and does not require the DB2 Runtime Client. However, in this mode, when the DB2 Content Manager application uses the JDBC driver, you must explicitly specify the host name and the port of the database that is being connected to. Specifying the host name and port eliminates the need to use the database catalog. However, if the host name, port, or the remote database are not specified in the `cmbicmsrvs.ini` file for ICM connector and in the `cmbds.ini` file for federated connector, the type 2 connection is used by default. The type 2 connection requires the DB2 Runtime Client and a database cataloged on the local system.

Important: If the host name, port, and the remote database name are specified, but the values are incorrect, an error results when DB2 Content Manager attempts to make a type 4 connection. To correct this error, you must update the INI file to include the correct information.

Important: If you upgrade to DB2 Content Manager Version 8.4, you can update the host name, port, and the remote database in the `cmbicmsrvs.ini` file for ICM connector and `cmbds.ini` file for the federated connector to use the type 4 driver.

The JAR files that are required for IBM DB2 Driver for JDBC and SQLJ (type 2 and type 4) are:

- `db2jcc.jar`
- `db2jcc_license_cu.jar`

These JAR files are packaged with DB2 Content Manager. An additional JAR file, db2jcc_license_cisuz.jar is required to connect to zSeries® servers. The db2jcc_license_cisuz.jar is also packaged with DB2 Content Manager.

Important: The old file (db2java.zip) is not used anymore and can be removed from the class path of any scripts.

Modifications to cmbicmsrvs.ini and cmbds.ini files

Your existing cmbicmsrvs.ini file for ICM connector and cmbds.ini file for federated connector supports IBM DB2 Driver for JDBC and SQLJ.

DB2 Content Manager also provides the ability to override the host name, port, and remote database name values by specifying the JDBC driver and JDBC URL directly. To specify the JDBC driver and the JDBC URL directly, DB2 Content Manager Version 8.4 provides two properties, ICMJDBCDRIVER and ICMJDBCURL in the cmbicmsrvs.ini file for ICM connector and cmbds.ini file for federated connector. You can also provide the JDBCURL and JDBC DRIVER properties by using the **connect_string** parameter associated with the DKDatastoreICM::connect method.

In DB2 Content Manager, these properties are optional and do not have any values. You can specify the values of these properties by choosing either of the following ways:

- If the ICMJDBCDRIVER property is not specified or is empty, the IBM DB2 Driver for JDBC and SQLJ (type 2 and type 4) driver is used by default. The new IBM DB2 Driver for JDBC and SQLJ (type 2 and type 4) driver is loaded through the class, com.ibm.db2.jcc.DB2Driver .
- If the ICMJDBCURL property is specified and the ICMHOST, ICMPORT, and ICMREMOVEDB information is also specified in the cmbicmsrvs.ini file and in the cmbds.ini file, the value associated with the ICMJDBCURL property takes precedence and is used as the URL to connect to the database.
- If the JDBCURL property is specified by using the **connect_string** parameter of the connect() method and the ICMJDBCURL property is specified in the cmbicmsrvs.ini file and in the cmbds.ini file, the JDBCURL property in the **connect_string** parameter takes precedence. Similarly, the JDBC DRIVER property passed through the **connect_string** parameter takes precedence over the ICMJDBCDRIVER property specified in the cmbicmsrvs.ini file and in the cmbds.ini file.

Setting both the URL and the driver is not a requirement in DB2 Content Manager Version 8.4. You might specify either the URL or the driver.

To specify a driver other than the IBM DB2 Driver for JDBC and SQLJ, the values for the ICMJDBCDRIVER and ICMJDBCURL properties must be manually set in the cmbicmsrvs.ini and cmbds.ini file. In Version 8.4, DB2 Content Manager supports only the IBM DB2 Driver for JDBC and SQLJ when connecting to a DB2 database. DB2 Content Manager does not support any other driver for a DB2 database.

You can set the values for the ICMJDBCDRIVER and ICMJDBCURL properties in the INI file as shown:

```
ICMJDBCDRIVER=com.ibm.db2.jcc.DB2Driver
ICMJDBCURL=jdbc:db2://myhostname.ibm.com:50000/icm1sdb
```

Remember:

When an IP address is used rather than a hostname in the JDBC URL, [and] are required for the address if IPv6 is used. For example, the connect string for an IPv6 address 2007::9:181:141:150 must be enclosed with [and] in the following format:

```
JDBCURL=jdbc:db2://[2007::9:181:141:150]:50000/icmn1sdb
```

Important: The INI file properties and the **connect_string** parameter applies to DB2.

Related information



Supported drivers for JDBC and SQLJ

Working with the DB2 Content Manager samples

DB2 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 DB2 Information Integrator for Content installation in order to have the samples in the directory.

Note: 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 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 8.4 functionality are grouped into what is called the ICM connector.

The ICM connector APIs have an ICM suffix, as in the example DKDatastoreICM.

Understanding the software components

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, and 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

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.

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.

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, "");
```

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 extracted.

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). Table 8 on page 86 contains the pre-defined item type classifications and their corresponding ID constant.

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 DKConstant.DK_CM_DKPARTS.
Part	DK_ICM_ITEMTY PE_CLASS_DOC_ PART	3	Items (parts) in the document model classification.

Notes: Note: Constants are located in com\ibm\mm\sdk\common\DKConstantICM.java for Java and dk\icm\DKConstantICM.h for C++.

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

1. Create an item type and pass it a reference to the content server.
2. Give the item type a name. 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.

Examples

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 list item types, you must first connect to a content server.

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.

Examples

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 SItemTypeInfoRetrievalICM sample.

Limitations to updating item types

There are some limitations to updating item types.

The following is the list of limitations to updating item types:

- You cannot update an item type's name. For example, if you define an item type with the name Book, then you cannot update this item type's name to BookNew.
- You cannot change an item type's identification ID. For example, if you define an item type Book and its ID is 1000, you cannot update Book's ID to 1002.
- You cannot update an item type's classification. For example, if you define an item type Book, and its classification is DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM, you cannot update it to DK_ICM_ITEMTYPE_CLASS_DOC_MODEL.
- The document part item type cannot be autolink enabled, neither during the creation nor during the update time.
- You cannot remove attributes or attribute groups.
- You cannot change attributes from required to non-required or non-required to required.
- You cannot change attributes from unique to non-unique or non-unique to unique.
- You cannot change an attribute's description. For example, if you define an attribute Address, and its description is, "This is an address attribute", you cannot update the attribute description to "This is a new address attribute".
- You cannot change a component type delete rule.
- You cannot change text searchable from true to false.
- In the text searchable field, you cannot change the value of IndexCCSID and IndexLangCode. For example, if you set the IndexCCSID value to 806 and IndexLangCode to EN_US, you cannot change the IndexCCSID value to 852 and indexLangCode to EN_AU.
- You cannot change the Item Type XDO classification. For example, if you define an item type Book, Book is a resource item type and its XDO classification ID is DK_ICM_XDO_LOB_CLASS_ID. You cannot change it to DK_ICM_XDO_TEXT_CLASS_ID.
- You cannot change the attribute length for VARCHAR, CHAR, BLOB, and CLOB. The total and the fixed places for decimal length cannot be changed.
- You cannot remove a child component type if it already has data loaded.

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. After 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. For further information about creating attributes, see the `SItemTypeCreationICM` sample.

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	<code>DKConstant.DK_CM_BLOB</code>	<code>byte bytes[]</code>
Char	<code>DKConstant.DK_CM_CHAR</code>	<code>java.lang.String</code>
CLOB	<code>DKConstant.DK_CM_CLOB</code>	<code>java.lang.String</code>
Date	<code>DKConstant.DK_CM_DATE</code>	<code>java.sql.Date</code>
Decimal	<code>DKConstant.DK_CM_DECIMAL</code>	<code>java.math.BigDecimal</code>
Double	<code>DKConstant.DK_CM_DOUBLE</code>	<code>java.lang.Double</code>
Integer	<code>DKConstant.DK_CM_INTEGER</code>	<code>java.lang.Integer</code>
Short	<code>DKConstant.DK_CM_SHORT</code>	<code>java.lang.Short</code>
Time	<code>DKConstant.DK_CM_TIME</code>	<code>java.sql.Time</code>
Timestamp	<code>DKConstant.DK_CM_TIMESTAMP</code>	<code>java.sql.Timestamp</code>
Varchar	<code>DKConstant.DK_CM_VARCHAR</code>	<code>java.lang.String</code>

3. Add the new attribute definition to the persistent content server.

Examples

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.

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.

The following example demonstrates how to create an attribute group:

Example: 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();

```

Example: 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);

```

For more information, see the SAttributeGroupDefCreationICM sample.

Listing the attributes in a content server

The example demonstrates how to get a list of attributes in a content server for Java and C++.

Example: Java

For Java

```

DKDatastoreICM dsICM;
DKDatastoreDefICM dsDefICM=DKDatastoreDefICM(dsICM.datastoreDef());
//Get a collection containingall Attribute Definitions.
DKSequentialCollection attrDefColl =
    (DKSequentialCollection)dsDefICM.listAttrs();
if ((attrDefColl!=null) &&(attrDefColl.cardinality()>0)){
    //Create an iterator to iterate through the collection
    dkIterator iter = attrDefColl.createIterator();
    while(iter.more()){
        //while there are still items in the list,continue
        dkAttrDef attrDef = (dkAttrDef) iter.next();
        System.out.println("-"+attr.getName()+":"+attr.getDescription());
    }
}

```

Example: C++

For C++

```

DKDatastoreICM * dsICM; .....
DKDatastoreDefICM*dsDefICM =(DKDatastoreDefICM *)dsICM->datastoreDef();
//Get a collection containingall Attribute Definitions.
DKSequentialCollection*attrDefColl =
    (DKSequentialCollection *)dsDefICM->listAttrs();
if (attrDefColl &&(attrDefColl->cardinality()>0)){

```



```

//Create an iterator to iterate through the collection
dkIterator*iter =attrDefColl->createIterator();
while(iter->more()){
    //while there are still items in the list,continue
    dkAttrDef* attrDef =(dkAttrDef *)iter->next()->value();
    cout <<"-"<<attrDef->getName()<<":"<<attrDef->getDescription()<<endl;
    delete(attrDef);
}
delete(iter);
delete(attrDefColl);
}
delete(dsDefICM);

```

Listing attribute names for an item type

The following example demonstrates how to get a list of attribute names for an item type.

Example: 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());
}

```

Example: 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);

```

For more information, see the SItemTypeRetrievalICM API education sample.

Creating an item

An item is the entire tree of component DDOs, with at least one root component.

An item or DDO must have a generic type property associated with it. This generic type of the DDO is a document, folder, or an item. When creating items, the item property is specified through the second parameter of the datastore's createDDO function mixed in with the semantic type. You submit only one value, but the item property and semantic type values are derived and set separately in the DDO. The item property types are a subset of semantic types. This value is stored in the

DDO's property, `DK_CM_PROPERTY_ITEM_TYPE`, which should not be confused with the overall item type definition that describes this the structure of this item. `PROPERTY_ITEM_TYPE` refers to the item property, not the item type definition which fully describes the item structure and settings.

Table 10 shows a list of available item property types.

Table 10. Item type property definitions

Property type	Constant	Definition
Document	<code>DK_CM_DOCUMENT</code>	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 because it does not rigidly mean an implementation of a specific document model. Setting a document type does not necessarily mean a document with parts using the DB2 Content Manager V8 document model which is instead controlled by the "document" classification in the item type definition.
Folder	<code>DK_CM_FOLDER</code>	Item represents a folder that can contain other items. This item features a built-in DKFolder collection which can hold folder contents which form folder links to other items. An empty DKFolder collection is automatically added. All items with this value are capable of using the DKFolder mechanism. However, this value does not necessarily mean that a solution uses the DKFolder mechanism and can instead implement its own folder solution. This value means any interpretation of a folder, but the built-in DKFolder is available to such items. Note: You can create non-folder links and references from items of any semantic type or item property type.
Item (default)	<code>DK_CM_ITEM</code>	Generic item. This is the item property type for all semantic types that are not document or folder.

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. DDO Constructors are not recommended. Using the various `DKDatastoreICM::createDDO()` methods are recommended ways to create DKDDO instances (or subclasses). Creating DDOs using `new` or any means of instantiating a DDO instance directly from a DDO class is not recommended. The `createDDO()` methods instantiate the correct sub class and set up any other necessary information within the DDOs to work correctly with other operations.

When creating items, a semantic type can be specified as the second parameter of the datastore's `createDDO` function. A semantic type in general is a metaphor for an item. Semantic types are primarily a feature for application use and are typically not enforced or especially meaningful within the DB2 Content Manager server. This gives items an enforceable characteristic for some basic operations and provides applications with the ability to apply a semantic type. Where applicable,

the semantic types are enforced within DB2 Content Manager, but primarily the only semantic type that has any meaning, validation, or enforcement is FOLDER.

Remember: Do not depend on semantic type values if your application reads items created by other applications or tools because the meaning and enforcement can vary among applications. For example, do not depend on the various part semantic types because parts are allowed to be set with any semantic type value. Instead, depend first on the PID's object type which identifies the exact item type or you can retrieve the definition to tell you more, such as classification as a part, XDO class, etc. You can depend on the semantic type as a secondary measure if you do not recognize the object type.

The value is stored in the DDO's property, DKConstantICM.DK_ICM_PROPERTY_SEMANTIC_TYPE, which in many cases may contain the same value as the item property type. Table 11 lists the available semantic types.

Table 11. Pre-defined semantic types

Semantic type	Constant	Definition
Any	User-specified	CreateDDO will assign any value specified to the semantic type value. Because applications can enforce semantics as they want, no values are disallowed, except for item types classified as "document" (document model).
Document	DK_ICM_SEMANTIC_TYPE_DOCUMENT	Item generically represents a document. Setting semantic type on a document does not necessarily mean a document with parts using the DB2 Content Manager V8 "document model" which is instead controlled by the "document" classification in the item type definition. Equivalent to item property type DK_CM_DOCUMENT.
Folder	DK_ICM_SEMANTIC_TYPE_FOLDER	Item represents a folder that can contain other items. This item features a built-in DKFolder collection which can hold folder contents which form folder links to other items. An empty DKFolder collection is automatically added. All items with this value are capable of utilizing the DKFolder mechanism. However, this value does not necessarily mean that a solution utilizes the DKFolder mechanism and could instead implement its own folder solution. This value means any interpretation of a folder, but the built-in DKFolder is available to such items. Note that you can create non-folder links and references from items of any semantic type or item property type. Equivalent to item property type DK_CM_FOLDER.
Container	DK_ICM_SEMANTIC_TYPE_CONTAINER	Item represents a generic container or generic item. It is equivalent to item property type DK_CM_ITEM. Solutions can implement a containment relationship through any solution of their choice except for the built-in DKFolder mechanism (folder links). You can use links of any other link type, reference attributes, or any other custom solution of your choice.

Table 11. Pre-defined semantic types (continued)

Semantic type	Constant	Definition
Base	DK_ICM_SEMANTIC_TYPE_BASE	Alleged to most nearly resemble document part of type "BASE" for use in the built-in document model where parts are created and added to make up a document. However, this value does not uniquely identify or make this a BASE part. This value is not validated or enforced. The item type of the part (identified in the PID's object type for the part DDO) is what makes this part a BASE part. The BASE parts can be created with any semantic type of your choice. Set this semantic type for BASE parts, but when detecting part types, use the PID's object type first. Use this value if you created your own custom part item types so that other applications that might not know your custom part type and could map it to the nearest similar part type that it understands.
BaseText	DK_ICM_SEMANTIC_TYPE_BASETEXT	Same as BASE (See "Base" documentation for details) except alleged to most nearly resembles "BASETEXT".
BaseStream	DK_ICM_SEMANTIC_TYPE_BASESTREAM	Same as BASE (See "Base" documentation for details) except alleged to most nearly resembles "BASESTREAM".
Annotation	DK_ICM_SEMANTIC_TYPE_ANNOTATION	Same as BASE (See "Base" documentation for details) except alleged to most nearly resembles "ANNOTATION".
History	DK_ICM_SEMANTIC_TYPE_HISTORY	Same as BASE (See "Base" documentation for details) except alleged to most nearly resembles "HISTORY".
Note	DK_ICM_SEMANTIC_TYPE_NOTE	Same as BASE (See "Base" documentation for details) except alleged to most nearly resembles "NOTE".
User defined	User defined	A user defined semantic type. Refer to the Semantic Type Samples for more information.

Note:

- In Java, the constants are defined in DKConstantICM.java.
- In C++, the constants are defined in DKConstantICM.h.

When you create an item, you create 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.

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 example demonstrates how to set and retrieve item attribute values.

Example: 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.

Example: 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")));
```

Setting and retrieving foreign key attribute values

In DB2 Content Manager Version 8, 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 provides methods in the `DKForeignKeyDefICM` 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.

Example: Java

```
ArrayList getSpecificAttrsForForeignKey(DKNVPair[] fkeyValues,  
String[] columnNames ) throws Exception, DKException
```

Returns an ArrayList of String arrays.

Example: 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

Table 12. Example results for parameters passed into *getSpecificAttrsForForeignKey()* (continued)

DKNVPairs	Attribute names	Result
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.

Example: Java

```
public Vector listForeignKeyAttrDetails( ) throws Exception, DKException
```

Example: C++

```
dkCollection* listForeignKeyAttrDetails()
```

The following examples define new associations between attributes of two component types into a foreign key definition.

Example: Java

```
public void addSrcAndTgtAttrName(String srcAttrName, String tgtAttrName, int columnSeq, boolean displayFlag) throws Exception, DKException
```

Example: 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.

Example: Java

```
public void setDisplayFlag(int columnSeq, boolean displayFlag) throws Exception, DKException
```

Example: 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.

Example: Java

```
public boolean getDisplayFlag(int columnSeq) throws Exception, DKException
```

Example: 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

Set setData method to the required value by specifying that value using the java.lang.Object, as shown in the example below.

Example: Java

In the example below, nameOfAttrStr is the string name of the attribute and valueObj is the value.

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

Example: C++

```
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,"book_title"),
    DKString("More Effective C++"));
```

Updating an item

To update an item, you must retrieve the item, modify its attributes, and call the update function.

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.

Example: Creating and updating a version

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 . For more information on updating items, see the `SItemUpdateICM` API education sample.

Defining a resource item type

A *resource item* is an item with additional system defined attributes that define the location, type, size, and so forth, of the object that the item represents.

The object is sometimes called a resource and can be a video file, an image, a word processor document, and so forth. For additional information, see the `SItemTypeCreationICM` sample (in the samples directory).

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

1. Get the content server definition object from the connected content server.

Java

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

C++

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*)  
dsICM->datastoreDef();
```

2. Create a new item type definition.

Java

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

C++

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

3. Add and 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 a resource item. Remember that there are multiple ways to set and store resource content.

1. Create the resource item.

Important: The resource item can be any semantic type and that the `DKDatastoreICM::createDDO` call is also used to create a resource item in the same way that it is used to create a regular item.

The type of resource returned from `DKDatastoreICM.createDDO` is cast to the correct sub-class (in this case `DKLobICM`) based on the XDO classification 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 `SResourceItemMimeTypeICM` sample.

3. Add the data to the content server. In this case, a sample Word document.

Remember: The resource item content is stored in the resource manager.

Java

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

C++

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

Important: In C++, you must clean up the memory.
`delete(lob);`

For more information, see the `SItemUpdateICM` sample.

The original name, name the item before it was stored in DB2 Content Manager, of a resource item or document part is stored in the library server as an attribute of the resource item or document part. To retrieve the original file name (if not already retrieved or if you want to refresh the value), call `retrieve()` (single-item or multi-item) and request options that at least include the system resource attributes, such as `DKRetrieveOptionsICM::baseAttributes(true)`. You might optionally add a filter for only system attributes (`DKRetrieveOptionsICM::attributeFilters()`). However, to refresh or retrieve system resource attributes, you do not need to retrieve the resource content. For example, you can use `DKRetrieveOptionsICM::resourceContent(false)` (default).

Searching for items

You can search the items that match your criteria.

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.
3. Save the query results in a `DKResult` object.

Example

Java

```
DKNVPair options[] = new DKNVPair[3];  
options[0] =  
new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "0"); // No Max (Default)
```

```

DKRetrieveOptionsICM dkRetrieveOptions =
DKRetrieveOptionsICM::createInstance(dsICM);
dkRetrieveOptions.baseAttributes(true);
options[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,
dkRetrieveOptions);
options[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
DKResults results = (DKResults)dsICM.evaluate(query,
DKConstantICM.DK_CM_XQPE_QL_TYPE, options);

```

C++

```

DKNVPair *options    = new DKNVPair[3];
// only one result will be returned
options[0].set(DK_CM_PARM_MAX_RESULTS, "1");// Retrieve only one item
DKRetrieveOptionsICM* dkRetrieveOptions =
DKRetrieveOptionsICM::createInstance(dsICM);
dkRetrieveOptions->baseAttributes(TRUE);
options[1].set(DK_CM_PARM_RETRIEVE, dkRetrieveOptions);
// Last option has to be this value
options[2].set(DK_CM_PARM_END ,(long)0);
//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);

```

Query timeout option

The query timeout option, `DKConstant.DK_CM_SQL_TIMEOUT`, can be added as an `DKNVPair` in the query options array. It takes an integer object representing the number of milliseconds before the query is timed out or cancelled. This option is supported in all the `DKDatastoreICM` query methods: `execute`, `evaluate`, `executeCount`, and `executeWithCallback`.

Important: This option is on a per-query basis, and must be passed with each DB2 Content Manager API query request.

See the following Java™ code snippet for an example to set the timeout option and to call the query method. The timeout option is available for both Java and C++ APIs.

```

// Create the query timeout option.
//Set to 5 seconds (5000 milliseconds)
DKNVPair options[] = null;
options = new DKNVPair[2];
options[0] = new DKNVPair( DKConstant.DK_CM_SQL_TIMEOUT,
new Integer(5000));
options[1] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
// Use one of the query methods to perform the query
dkResultSetCursor cursor = dsICM.execute(
"/NOINDEX", DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
DKResults results = (DKResults) dsICM.evaluate(
"/NOINDEX", DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
dsICM.executeWithCallback(
"/NOINDEX", DKConstantICM.DK_CM_XQPE_QL_TYPE, options, callbackObject);
long count = dsICM.executeCount(
"/NOINDEX", DKConstantICM.DK_CM_XQPE_QL_TYPE, options);

```

For additional information, see the `SSearchICM` sample.

Retrieving an item

DB2 Content Manager provides several retrieve options.

To retrieve an item, you must first obtain a DDO object with completed PID information that represents the item (or the component). If you have access to a completed PID object or know the PID string of the item to be retrieved, you can

reconstruct a DDO object by using the `DKDatastoreICM::createDDOFromPID()` method. The `DKDatastoreICM::createDDOFromPID()` method accepts a PID object or a PID string. You can also use a query to find the items and return DDOs with completed PID information.

Note: The query determines the list of PIDs and creates blank DDOs with the completed PIDs, but the query can call a multi-item retrieve method before returning the DDOs based on your query options.

After you have obtained any valid DDO object, you can retrieve or refresh its contents. You can directly retrieve any DDO that includes the root component, the child component, and individual document part DDOs. An item is accessed through the root component DDO.

There are two types of retrieve methods: multi-item retrieve and single-item retrieve. Multi-item retrieve operation offers a significant performance advantage over single-item retrieve operation that makes separate calls to the server. Whenever you plan to retrieve more than one DDO, you must always use the multi-item retrieve operation unless you require data that can be retrieved only through the single-item retrieve operation.

For example, resource content can be retrieved only through the single-item retrieve operation. When you specify the retrieve options in a query (or search), the query uses the multi-item retrieve operation on the results. You should use the multi-item retrieve operation for all common metadata required from all DDOs. When one DDO requires additional content, such as resource content for a specific item, that item should be retrieved with the single-item retrieve method. For examples of method signatures of single-item retrieve and multi-item retrieve operations and query, see `DKDatastoreICM::retrieveObjects(dkCollection,DKNVPair[])` in *Application Programming Reference*.

DB2 Content Manager provides a `DKRetrieveOptionsICM` class and an integrated `DKProjectionListICM` class. The granular retrieve options are designed to select data that is required. In addition to the built-in optimizations automatically enabled by using `DKRetrieveOptions` method, you can get better retrieval performance by retrieving only the subset of data that you plan to use.

Retrieving child components and document parts

You can directly retrieve or refresh individual child components and document part DDOs without calling the retrieve operation on the parent DDOs. However, you cannot update the items by retrieving the component and calling the `DKDDO.update()` method. All updates must be done through the root component for the item or document. You must modify the child component or document part within the retrieved structure of the root component before calling the `DKDDO.update()` method on the root component.

Specifying requests and exclusions by using retrieve options

Always specify your list of requests and exclusions for a better performance based on the performance considerations that are documented in the `DKRetrieveOptionsICM` class under *Application Programming Reference*. Submitting a `DKRetrieveOptionsICM` instance enables additional retrieve optimizations that are not available through default or bitwise option behavior if no `DKRetrieveOptionsICM` instance is submitted.

Important: If you do not specify requests and exclusions, the application performance can be severely impacted.

You must use retrieve options specified through `DKRetrieveOptionsICM` class to specify what subset of data is required. You should retrieve only the data that you use. You can improve your performance by using retrieve options carefully and spending performance costs only for the data that you use.

For data that you might use for one item only, defer additional retrievals until that data is required. For example, if you are listing 10 000 query results, consider retrieving only minimal attributes for the first page of results and retrieve more data only if the user requests the next page or clicks on a document to view.

You can avoid unnecessary calls to the server, network communication, memory use and object counts for unnecessary objects, and other overhead. Performance implications of every retrieve option setting is documented in the Javadoc reference documentation in `DKRetrieveOptionsICM` class.

You must always provide retrieve options even if you do not retrieve any new data. The retrieve options that you set are built incrementally with nothing selected by default. Each retrieve option that you set as true is considered a specific request for data and any option that is set as false (default) is considered a specific request to exclude that data. It is better to submit an options object with no changes to demonstrate that you are not requesting any data. If retrieve options are not specified, various selections based on the deprecated bitwise integer retrieve options variants are included by default. The default varies depending on which method you use.

Important: Always use `DKRetrieveOptionsICM` class whenever possible over the bitwise integer or long retrieve options or no-option variants. By using `DKRetrieveOptionsICM` class, you enable a number of optimizations and product updates that are not available when you use the old bitwise integer retrieve options, even with equivalent retrieve settings. If you had previously used the bitwise integer retrieve options, see the `DKRetrieveOptionsICM` class under *Application Programming Reference*.

For more details, see the Javadoc information for:

- `DKRetrieveOptionsICM`
- `DKDatastoreICM::retrieveObjects(dkCollection, DKNVPair[])`

Additional documentation and examples are provided in the `SItemRetrievalICM` sample.

Retrieving an item: Java:

1. After connecting using a `DKDatastoreICM` object, search for the item by using the appropriate item ID or other attribute conditions.
2. Save the query results in a `DKResults` object if you are using `evaluate()`. In the code example below, `queryStr` is the search string in the correct query language format.

```
DKRetrieveOptionsICM dkRetrieveOpts =
DKRetrieveOptionsICM::createInstance(dsICM);
DKNVPair options[] = new DKNVPair[2];
options[0] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,dkRetrieveOpts);
options[1] = new DKNVPair(DKConstant.DK_CM_PARM_END,null);
DKResults results = (DKResults)dsICM.evaluate(queryStr,
DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
```


3. Create an iterator on DKResults and use it to get DDOs, with completed PIDs
4. Instead of using the query, if you have a PID string that you obtained from a DDO previously and use that PID string to reconstruct a new DDO for an existing item, use the DKDatastoreICM's createDDOFromPID(DKPidICM pid) or createDDOFromPID(String pidString) methods as shown in the example below. Do not use the DKDDO constructor. You can use this method to recreate a blank DDO if you want to completely clear the existing data within an existing DDO.

Note: If you just performed a query, you can use the DDOs returned from query directly.

```
DKPidICM  pid    = ddo.getPidObject();
String    pidStr = pid.pidString();
DKDDO     ddo    = dsICM.createDDOFromPID(pid);
DKDDO     ddo2   = dsICM.createDDOFromPID(pidStr);
```

After you have a DDO from Step 4 that you want to retrieve, if you want to retrieve root attributes, you can use a single-item retrieve operation.

```
DKRetrieveOptionsICM dkRetrieveOpts =
DKRetrieveOptionsICM.createInstance(dsICM);
dkRetrieveOpts.baseAttributes(true);
ddo.retrieve(dkRetrieveOpts.dkNVPair());
```

Remember: Use multi-item retrieve for more than one DDO.

Alternatively, you can use multi-item retrieve for retrieving multiple DDOs and reuse the same options from the last code example.

```
dkCollection ddoColl = new DKSequentialCollection();
ddoColl.addElement(ddo);
dsICM.retrieveObjects(ddoColl, dkRetrieveOpts);
```

If an item is enabled for versioning, you can retrieve a specific version of the item by using query with conditions on the version ID or by modifying the version number in the PID prior to retrieval. You can also request the latest version as shown below.

```
DKRetrieveOptionsICM dkRetrieveOpts =
DKRetrieveOptionsICM::createInstance(dsICM);
dkRetrieveOpts.baseAttributes(true);
dkRetrieveOpts.functionVersionLatest(true);
ddo.retrieve(dkRetrieveOpts.dkNVPair());
```

For single-item retrieve, you can also call the datastore single-item retrieve method directly. This is the actual single-item retrieve method that all other single-item retrieve methods ultimately call.

```
dsICM.retrieveObject(ddo, dkRetrieveOpts);
```

Retrieving an item: C++:

1. After connecting using a DKDatastoreICM object, search for the item by using the appropriate item ID or other attribute conditions.
2. Save the query results in a DKResults object if you are using evaluate(). In the code example below, queryStr is the search string in the correct query language format.

```
DKNVPair* options = new DKNVPair[2];
options[0]->set(DK_CM_PARM_RETRIEVE, dkRetrieveOpts);
options[1]->set(DKConstant.DK_CM_PARM_END, (long)0);
DKResults* results = (DKResults*)(dkCollection*)
dsICM->evaluate(queryStr, DK_CM_XQPE_QL_TYPE, options);
```


3. Create an iterator on DKResults and use the iterator you created to get DDOs with completed PIDs.
4. Instead of using query, if you have a PID string that you obtained from a DDO previously and you want to use the PID string to reconstruct a new DDO for an existing item, use the DKDatastoreICM's createDDOFromPID(DKPidICM pid) or createDDOFromPID(String pidString). Do not use the DKDDO constructor. You can use this method to recreate a blank DDO if you want to completely clear the existing data within an existing DDO.

Note: If you just performed a query, you can use the DDOs returned from query directly.

```
DKPidICM* pid    = (DKPidICM*) ddo->getPidObject();
DKString  pidStr = pid->pidString();
DKDDO*    ddo    = dsICM->createDDOFromPID(pid);
DKDDO*    ddo2   = dsICM->createDDOFromPID(pidStr);
```

After you have a DDO from Step 4 that you want to retrieve, if you want to retrieve root attributes, you can use a single-item retrieve operation.

```
DKRetrieveOptionsICM* dkRetrieveOpts =
DKRetrieveOptionsICM::createInstance(dsICM);
dkRetrieveOpts->baseAttributes(TRUE);
ddo->retrieve(dkRetrieveOpts->dkNVPair(), dkRetrieveOpts->dkNVPairLength());
```

You might want to use multi-item retrieve for retrieving multiple DDOs and reuse the same options from previous code example.

```
dkCollection* ddoColl = new DKSequentialCollection();
ddoColl->addElement(ddo);
dsICM->retrieveObjects(ddoColl, dkRetrieveOpts);
```

If an item is enabled for versioning, you can retrieve a specific version of the item by using query with conditions on the version ID or by modifying the version number in the PID prior to retrieval. You can also request the latest version as shown below.

```
DKRetrieveOptionsICM* dkRetrieveOpts =
DKRetrieveOptionsICM::createInstance(dsICM);
dkRetrieveOpts->baseAttributes(TRUE);
dkRetrieveOpts->functionVersionLatest(TRUE);
ddo->retrieve(dkRetrieveOpts->dkNVPair(), dkRetrieveOpts->dkNVPairLength());
```

For single-item retrieve, you can also call the datastore single-item retrieve method directly. This is the actual single-item retrieve method that all other single-item retrieve methods ultimately call.

```
dsICM->retrieveObject(ddo, dkRetrieveOpts);
```

Deleting items

Use the delete method in the DDO to delete an item from the content server.

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

Example: Java

```
ddoDocument.del();
```

Example: C++

```
ddoDocument->del();
```

For more information about deleting items, see the `SItemDeletionICMAPI` 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 or lock 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, child components, resource content, and document parts. Linked and referenced items, however, are not locked with the item because the items are only referenced, not owned. The persistent write-lock on the item is released when you check in the item.

The check-out and check-in operations are equivalent to lock and unlock respectively. These operations do not retrieve the latest information upon checkout and do not persist updates upon check in except when passed as options to retrieve or update operations. Therefore, when you check out (lock) an item, remember that any other user or connected session could have made changes to the item between the time you last retrieved and when you obtain the lock. You might either refresh the `DDO` with the latest information after you have obtained the lock (after which guarantees you are the only user that can make a change) or request checkout as part of the original retrieve operation using retrieve options.

The write-lock is persistent and tied to your user name and not to your connected session. The locks remain until you or a super-user with sufficient authority unlocks your item. Furthermore, the same user is allowed to make changes in any authenticated session. If you connect with the same user name to multiple connections, all have permissions to the write-lock and can make changes.

You can retrieve the checked out status which indicates the current lock owner (if any) and the timestamp of the lock through retrieve with option, `DKRetrieveOptions::basePropertyCheckedOutDetails(Boolean)`. See the detailed information in the *Application Programming Reference* for this method for

information on usage, behavior, and which DDO properties the information is stored in. You can also use DKDatastoreExtICM methods for making a separate call to check the current checked-out status and find the current lock owner, but at the cost of a server call per check.

To check individual items in and out, use the DKDatastoreICM's checkIn and checkOut operations, as shown in steps below. You can combine check out with retrieve or combine check in with update as shown below.

1. If you have connected to a DKDatastoreICM object named dsICM, and using a DDO called myDDO, check out the item and check it back in.

Note: Normally you perform operations between check out and check in.

Java

```
dsICM.checkOut(myddo);  
dsICM.checkIn(myddo);
```

C++

```
dsICM->checkOut(myddo);  
dsICM->checkIn(myddo);
```

2. If you want to combine check out with retrieve and combine check in with update, follow the example below:

Note: Normally you perform operations between retrieve and update). If required, you can obtain the checked out status which will confirm your lock ownership.

Remember: Request information that you plan to use to minimize costs.

Java

```
DKRetrieveOptionsICM dkRetrieveOpts =  
    DKRetrieveOptionsICM.createInstance(dsICM);  
dkRetrieveOpts.baseAttributes(true);  
dkRetrieveOpts.basePropertyCheckedOutDetails(true);  
dkRetrieveOpts.functionCheckOut(true);  
ddo.retrieve(dkRetrieveOpts.dkNVPair());  
ddo.update(DKConstant.DK_CM_CHECKIN);
```

C++

```
DKRetrieveOptionsICM* dkRetrieveOpts =  
    DKRetrieveOptionsICM::createInstance(dsICM);  
dkRetrieveOpts->baseAttributes(TRUE);  
dkRetrieveOpts->basePropertyCheckedOutDetails(TRUE);  
dkRetrieveOpts->functionCheckOut(TRUE);  
ddo->retrieve(dkRetrieveOpts->dkNVPair(),  
    dkRetrieveOpts->dkNVPairLength());  
ddo->update(DK_CM_CHECKIN);
```

3. To check out (lock) multiple items at the same time, you can use multi-item retrieve with option DKRetrieveOptionsICM::functionCheckout(true). The following example reuses the same options from the Step 2.

Java

```
dkCollection ddoColl = new DKSequentialCollection();  
ddoColl.addElement(ddo1);  
ddoColl.addElement(ddo2);  
ddoColl.addElement(ddo3);  
dsICM.retrieveObjects(ddoColl, dkRetrieveOpts.dkNVPair());
```

C++

```
dkCollection* ddoColl = new DKSequentialCollection();
ddoColl->addElement(ddo1);
ddoColl->addElement(ddo2);
ddoColl->addElement(ddo3);
dsICM->retrieveObjects(ddoColl,
dkRetrieveOpts->dkNVPair(),dkRetrieveOpts->dkNVPairLength());
```

For more information, behavior notes, performance considerations, and more, see the *Application Programming Reference* for method `DKRetrieveOptionsICM::functionCheckOut(Boolean)`.

For more education, examples, and demonstrations on checking items in and out, see the `SItemUpdateICM` API education sample.

Setting and getting an item's versioning properties

The example below, demonstrates how to set an item's versioning properties and how to retrieve and item's versioning properties.

Example: Java

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();
// Accessing version information
String version = pid.getVersionNumber();
...
// Setting the version number in the DDO
pid.setVersionNumber(version);
```

Example: 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.

Example: Java

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

Example: C++

```
short versionControlPolicy = 0;
DKItemTypeDefICM * item    = NULL;
...
versionControlPolicy = item->getVersionControl();
```

Setting version control for an item type

Example: Java

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

Example: 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 `SItemTypeRetrievalICMAPI` education sample.

Getting the maximum number of versions allowed for an item type

Example: Java

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

Example: 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.

Example: Java

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

Example: C++

```
short versionMax    = 10;
DKItemTypeDefICM * item = NULL;
....
item->setVersionMax(versionMax);
```

Setting the value of the versioning type for an item type

Example: C++

```
DKItemTypeDefICM * item = NULL;
short versionType = DK_ICM_ITEM_VERSIONING_OPTIMIZED;
...
item->setVersioningType(versioningType);
```

Working with folders

A folder is like any other item created from any item type at run time.

When you create an item of semantic type folder, you get an attribute, DKFolder with a value of a DKFolder collection. This is called the folder mechanism. You can add DDOs of the folder contents to this DKFolder collection. The collection stores the root component DDO references to the items contained within the folder.

Important: The relationships of the folder to its contents are link relationships. Link relationships are not version-specific and does not mean ownership. Therefore, the relationship of the folder to its contents are not version-specific and the folder does not own its contents. The folder can support the data model of standard items, resource items, and documents depending on the item type classification.

Though the folder feature is flexible and works on any item type, it is common to set a separate item type for your folders. Combining both the folder mechanism and resource content or document parts is sometimes unnecessary and easier to manager as separate concepts. Folders are flexible and you can decide how much of flexibility you can use. The built-in folder mechanism works on items from any item type of any item type classification (non-resource, resource, and document model). However, most users prefer to have a separate item type that they can use within their application. Folders are flexible and you can decide how much of that flexibility you need.

If you prefer version-specific folder relationships or apply alternate delete rules to the relationship, you can model your own folder mechanism by modeling a child component type that contains a reference attribute which is version specific and can have alternate delete rules. The child collection acts as a multi-value attribute which allows you to have any number of reference attributes that point to your custom folder contents instead of using the built in DKFolder collection.

The procedures below include code fragments for informational purposes only.

Important: 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.

The item property type or semantic type must be specified as `DKConstant.DK_CM_FOLDER` when you are creating folder items. For more information, see the `SItemCreationICM` sample (in the samples directory).

A folder item is created by 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.

Example: Java

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

Example: C++

```
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);
```

Use `setData` method to populate the `ddoFolder`. After the folder DDO is created, it is saved to the persistent content server.

Example: Java

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

Example: C++

```
ddoFolder->add();
```

Adding contents to a folder

The `DKFolder` contents 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`. Adding multiple copies of the same item to `DKFolder` creates conflicting or duplicate modifications because all folder modifications are tracked.

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.

Adding contents to a folder example: Java:

To add contents to a folder:

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. Get 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. Check out the folder before updating it (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. Save the changes to the folder.  
`ddoFolder.update();`
  7. Check in the changes.  
`dsICM.checkIn(ddoFolder);`

### Adding contents to a folder example: C++:

To add the contents to a folder:

1. Create the items that will be added to the folder.  

```

DKDDO * ddoDocument = dsICM->createDDO("book", DK_CM_DOCUMENT);
DKDDO * ddoFolder2 = dsICM->createDDO("book", DK_CM_FOLDER);
DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);

```
2. Persist the created items to the datastore.  

```

ddoDocument->add();
ddoItem->add();
ddoFolder2->add();

```
3. Get 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 updated.  
`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 also checks in the folder (unlocks it).  
`ddoFolder->update();`
7. Check in the folder.  
`dsICM->checkIn(ddoFolder);`

### Removing contents from a folder

You can remove contents from a folder.

Items can be removed from a folder in one of following two ways:

#### Deferred removal

A deferred removal is done by using the `removeElementXX` method(s). The actual removal is done when the folder DDO is updated and multiple content server calls are combined into one.

#### Immediate removal

An immediate removal can be done by using the `dkFolder.removeMember` method. This removal is expensive because multiple calls are made to the content server.



See the SFolderICM sample (in the samples directory) for details on working with folders.

### Removing contents from a folder: Java:

To remove the contents from a folder:

1. Check out the folder DDO from which you want to remove an item.  
dsICM.checkOut(ddoFolder);
2. Look for the item to remove by creating an an iterator to iterate through the folder's content. In this case, look for the docItem DDO created earlier.

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

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

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

### Removing contents from a folder: C++:

1. Create an item and add it to a folder.

#### Note:

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 get good performance when you are adding and removing folder contents if you use element methods to group multiple updates and use single and periodic create and delete operations.

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 `addElement` or `removeElement` 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

When you are retrieving existing items, the DKFolder collection is not automatically included. You can request listing of folder contents which populates DKFolder collection and the latest list of contents.

**Important:** There is a difference between listing folder contents and retrieving data for the contents within a folder.

You must first list the folder content by populating the DKFolder collection with blank DDOs and their completed PIDs that represent the contents. After listing the folder contents, you can subsequently retrieve attributes or other information for the folder contents. You can pass the DKFolder collection directly to multi-item retrieve (`DKDatastoreICM::retrieveObjects(dkCollection,DKNVPair[])`) method because DKFolder collection is a `dkCollection` which contains DKDDOs.

To retrieve the latest list of folder contents, you must retrieve outbound links because folder relationships are link relationships. Optionally, you can restrict the link types that you retrieve. If you request the DKFolder link type only, you retrieve the DKFolder collection only and not other link types or link collections. If no folder contents are found, an empty DKFolder collection appears in the DDO. The following retrieve options apply to listing folder contents:

- `DKRetrieveOptionsICM::linksOutbound()`
- `DKRetrieveOptionsICM::linksTypeFilter()` (optionally restricts the type)
- `DKRetrieveOptionsICM::linksLevelTwoCount()`
- `DKRetrieveOptionsICM::linksLevelTwo()` (Java only method)

Important performance considerations, descriptions, and explanations of exactly where the data is retrieved into the DDO is fully explained in the programming reference for the retrieve options. See the `DKRetrieveOptionsICM::linksOutbound()` method in the *Application Programming Reference*.

**Important:** You must retrieve the semantic type property if you are retrieving the folder contents listing because folder functionality depends on semantic type folder. You can include attributes, or if you want, you can filter attribute retrieval to only system attributes. see `DKRetrieveOptionsICM::attributeFilters()` method in the *Application Programming Reference*.

The following example demonstrates how to retrieve the list of folder contents.

#### Example: Java

```
DKRetrieveOptionsICM dkRetrieveOpts =
DKRetrieveOptionsICM.createInstance(dsICM);
dkRetrieveOpts.baseAttributes(true);
dkRetrieveOpts.linksOutbound(true);
ddoFolder.retrieve(dkRetrieveOpts.dkNVPair());
```

#### Example: C++

```
DKRetrieveOptionsICM* dkRetrieveOpts =
DKRetrieveOptionsICM.createInstance(dsICM);
dkRetrieveOpts->baseAttributes(TRUE);
dkRetrieveOpts->linksOutbound(TRUE);
ddoFolder->retrieve(dkRetrieveOpts->dkNVPair(),
dkRetrieveOpts->dkNVPairLength());
```

### Obtaining all folders containing a specific DDO

Multiple folders can contain the same item (DDO) because folder relationships do not imply ownership. To determine which folders currently contain a specific DDO, the process is similar to retrieving and accessing the `DKFolder` collection, except that the folder sources are stored in a different collection and are requested using a different retrieve option.

For any root component DDO of an item, you can check to see if it is listed under any folders by retrieving the inbound links and requesting inbound folder sources. You can optionally restrict link retrieval to the `DKFolder` link type only if you want. The following retrieve options apply to listing folder sources that contain the current DDO:

- `DKRetrieveOptionsICM::linksInbound()`
- `DKRetrieveOptionsICM::linksInboundFolderSources()`
- `DKRetrieveOptionsICM::linksTypeFilter()` (optionally restricts the type)

To list the folder sources, you do not need to retrieve the semantic type of the current DDO unlike listing the folder contents. However, like the `DKFolder` collection, the folder sources collection contains blank DDOs with completed PIDs. You can subsequently call multi-item retrieve method and pass the `dkCollection` from the folder sources.

For more information on performance considerations, and details on exactly where the folder source collection is retrieved to (including all names and value types), see the `DKRetrieveOptionsICM::linksInboundFolderSources(Boolean)` method in *Application Programming Reference*.

The following examples demonstrate retrieving and accessing the list of folder sources that contain the current DDO. You can also use `DKDatastoreExtICM::GetFoldersContainingDDO()` method although the method is less efficient. See the `SFolderICM` sample (in the samples directory) for additional details about working with folders.

### Obtaining all folders containing a specific DDO: Java:

To obtain all folders containing a specific DDO:

1. If you have a DDO called ddo, request listing of folder sources that contain this DDO.

```
DKRetrieveOptionsICM dkRetrieveOpts =
DKRetrieveOptionsICM.createInstance(dsICM);
dkRetrieveOpts.linksInbound(true);
dkRetrieveOpts.linksInboundFolderSources(true);
dkRetrieveOpts.linksTypeFilter(DKConstantICM.DK_ICM_LINKTYPENAME_DKFOLDER);
ddo.retrieve(dkRetrieveOpts.dkNVPair());
```

2. Get the folder sources collection

```
dkCollection foundSourceFoldersColl = null;
propId = ddo.propertyId(DKConstant.DK_CM_PROPERTY_FOLDER_SOURCES);
if(propId > 0)
 foundSourceFolderColl = (dkCollection) ddo.getProperty(propId);
```

3. Examine results.

```
if(foundSourceFolderColl==null)
 System.out.println("No source folders found.");
else
 System.out.println("Found '"+foundSourceFolderColl.cardinality()
 +" source folders that contain this item.");
// Want to access the source folders?
dkIterator iter = foundSourceFolderColl.createIterator();
while(iter.more()){
 DKDDO folder = (DKDDO) iter.next();
}
```

### Obtaining all folders containing a specific DDO: C++:

To obtain all folders containing a specific DDO:

1. If you have a DDO called ddo, request listing of folder sources that contain this DDO.

```
DKRetrieveOptionsICM* dkRetrieveOpts =
DKRetrieveOptionsICM::createInstance(dsICM);
dkRetrieveOpts->linksInbound(TRUE);
dkRetrieveOpts->linksInboundFolderSources(TRUE);
dkRetrieveOpts->linksTypeFilter(DK_ICM_LINKTYPENAME_DKFOLDER);
ddo->retrieve(dkRetrieveOpts->dkNVPair(),dkRetrieveOpts->dkNVPairLength());
```

2. Get the folder sources collection

```
dkCollection* foundSourceFoldersColl = NULL;
propId = ddo->propertyId(DK_CM_PROPERTY_FOLDER_SOURCES);
if(propId > 0)
 foundSourceFolderColl = (dkCollection*) ddo->getProperty(propId);
```

3. Examine results

```
if(foundSourceFolderColl==NULL)
 cout << "No source folders found." << endl;
else
 cout << "Found '" << foundSourceFolderColl->cardinality() <<"'
 source folders that contain this item." << endl;
// Want to access the source folders?
std::auto_ptr iter(foundSourceFolderColl->createIterator());
while(iter->more()){
 DKDDO* folder = (DKDDO*)(dkDataObjectBase*)(iter->next());
}
```

## 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.

## DKLink definition example

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.

**Remember:** Before you make a call to the DDO add() or update() methods, all the items that are associated with links must already be persistent.

## 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 on page 120, 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/icm* or *IBMCMROOT/samples/java/icm* directory, for additional information about links. For information on creating user-defined link types, see the SLinkTypeDefinitionCreationICM sample.

## Retrieving linked items

There is a difference between listing links and retrieving data for the linked items. To retrieve attributes and other information, populate the DKLinkCollection with DKLinks and then call retrieve method. Multi-item retrieve method also supports link collections and you do not have to iterate through the link collection list and retrieve links one by one.

First, you must list the links by populating the DKLinkCollection collections with DKLinks, each of which contain blank DDOs and their completed PIDs that represent the linked source, target, and descriptor. Then, if you want, you can make a subsequent call to retrieve attributes or other information for the linked items themselves. You can pass or create your own dkCollection by creating a new DKSequentialCollection, iterate over all the links in all link collections wanted, and add other DDO (the current DDO reference appears as either the source or target depending on inbound or outbound direction), and then pass your new collection directly to multi-item retrieve (DKDatastoreICM::retrieveObjects(dkCollection,DKNVPair[]).

DKDatastoreICM::retrieveObjects(dkCollection) method takes any dkCollection of DKDDO objects. You do not have to iterate over your DKLinkCollection objects

retrieving source or target items one by one. However, you have to supply `DKDatastoreICM::retrieveObjects(dkCollection)` with a collection of `DKDDO` objects. Because a `DKLinkCollection` contains `DKLink` objects, you must create your own `dkCollection` (such as, by using the `DKSequentialCollection`) and insert the `DKDDO` objects that you want to retrieve. Then pass your `dkCollection` to multi-item retrieve method.

**Remember:** `DKLink` objects contain a reference to both the source and target item `DKDDO` objects.

Furthermore, one of the two `DKDDO` objects (source or target) is the same `DKDDO` as the item you already retrieved from which you obtained the link collections. You need to retrieve the non-retrieved `DKDDOs` only, such as the target of an outbound link or source of an inbound link. For more information on inbound and outbound links, refer to the `SLinksICM` API education sample.

There are many considerations and choices when it comes to retrieving the listing of links. There are very important performance considerations and full details documented for each links option. See the following links-related options for full details in the *Application Programming Reference*. You can retrieve only inbound, only outbound, both directions, limit to a specific link type, retrieve two levels of links at once (Java only), retrieve the links count for the second level, and more.

- `DKRetrieveOptions::linksOutbound()`
- `DKRetrieveOptions::linksInbound()`
- `DKRetrieveOptions::linksInboundFolderSources()`
- `DKRetrieveOptions::linksDescriptors()`
- `DKRetrieveOptions::linksTypeFilter()`
- `DKRetrieveOptions::linksLevelTwoCount()`
- `DKRetrieveOptions::linksLevelTwo()` (Java only)

---

## 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:

- Create privilege sets to group privileges together
- 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:



## Creating a privilege: 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());
```

## Creating a privilege: 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 steps in the following topics:

### Creating a privilege set: 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());
}
```

### Creating a privilege set: 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 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 demonstrates how to display a privilege set's properties.

Example: 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.

Defining an access control list (ACL): Java

1. Define a new DKACLData object. A DKACLData class is used to hold ACL related data.

```

DKACLData aclData1 = new DKACLData();

```
2. Set the user group name for the ACL

```

aclData1.setUserGroupName("ICMADMIN");

```
3. Set ACL patron type.

```

        aclData1.setPatronType(DK_CM_USER_KIND_USER);
4. Set the privilege set associated with this ACL
        aclData1.setPrivilegeSet(privSet_1);
5. Create another DKACLData object.
        DKACLData aclData2 = new DKACLData();
        aclData2.setUserGroupName("ICMPUBLIC");
        aclData2.setPatronType(DK_CM_USER_KIND_GROUP);
        aclData2.setPrivilegeSet(privSet_2);
6. Create a new ACL. A DKAccessControlListICM represents a CM V8. ACL
        DKAccessControlListICM acl1 = new DKAccessControlListICM(ds);
7. Assign a new name to the newly-created ACL
        acl1.setName("UserACL");
8. Assign a description to the newly-created ACL
        acl1.setDescription("This is a user-defined ACL");
9. Add the previously created ACL data objects to the ACL
        acl1.addACLData(aclData1);
        acl1.addACLData(aclData2);
10. Add the newly-created ACL to the authorization manager, aclMgmt.add(acl1);
11. Retrieve and display information about the newly created ACL
        DKAccessControlListICM acl_1 = (DKAccessControlListICM)
        aclMgmt.retrieveAccessControlList("UserACL");
        System.out.println("ACL name = " + acl_1.getName());
        System.out.println("    desc = " + acl_1.getDescription());
        dkCollection coll = acl_1.listACLData();
        dkIterator iter = coll.createIterator();
        while (iter.more()) {
            DKACLData aclData = (DKACLData) iter.next();
            DKPrivilegeSetICM _privSet = (DKPrivilegeSetICM)
                aclData.getPrivilegeSet();
            System.out.println(" PrivSet name = " + _privSet.getName());
            System.out.println(" PrivSet desc = " + _privSet.getDescription());
            String usrGrpName = aclData.getUserGroupName();
            System.out.println("    UserGroupName = " + usrGrpName);
            short patronType = aclData.getPatronType();
            System.out.println("    Patron type = " + patronType);
        }

```

The complete sample program is in the samples directory.

Defining an access control list (ACL): 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

You can retrieve and display ACL information by retrieving the ACL first and then retrieving the ACL data.

To retrieve and display ACL information, complete the following steps.

1. Retrieve the ACL from the authorization management object using the ACL's name. For 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. For 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, "
 cout<<" Patron type = "<< szpatronType << endl;
}
delete(iter);

```

## Assigning an ACL to an item type

After 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 a new 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 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 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.

### Assigning an ACL to an item: Java

1. Add a new property to the DDO. Assume that an ACL (Access Control List) named "MyACL" // already exists in the system. This property will be called DK\_ICM\_PROPERTY\_ACL.  

```
int propId =ddoItem.addProperty(DK_ICM_PROPERTY_ACL);
```
2. Set the previously created (or retrieved) ACL as the value of this property.  

```
ddoItem.setProperty(propId,"MyACL");
```
3. Persist the DDO into the data store.  

```
ddoItem.add();
```

## Assigning an ACL to an item: 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.

**Important:** 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();
```

## Using view ACLs on delete, add, or remove link operations

Item types can belong to one base view or more views.

Each item type view is assigned its own access control list (ACL). You can use view ACL only when the ACL binding is either item type or mixed with item-level ACL flag set to False. Therefore, to use view ACL, the ACL must be checked at the item-type view ACL instead of checking at the item-type ACL. If no view is specified, the base view ACL (which is the item-type ACL) is checked.

When you are adding or removing a link or deleting an item, the ACL on the view is checked. Therefore, to link to an item, users with an access to the item that you want to link from must also have an access to the item you want to link to.

If a user is part of an ACL (such as ACL 1) that gives the user read and write access to all items of an item type, the same user belongs to a second ACL (such as ACL 2) that gives the user only read access to all items of a item type view. If the user sends a query using the item type view and then performs a link action, the user cannot link or unlink the items because the item type view ACL is applied by the DB2 Content Manager system.

To link to the items that a user might not have access to, set a system flag bit in the system control table. The following sections demonstrate how to link to the items that you do not have access to by setting the system flag:

### Example: Java

```
DKDatastoreICM dsICM = new DKDatastoreICM();
DKDatastoreDefICM dsDef = null;
DKDatastoreAdminICM dsAdmin = null;
DKConfigurationMgmtICM dsCfg = null;
DKLSConfigurationMgmtICM lsCfg = null;
DKLSCfgDefICM lsDef = null;
short systemFlag = 0;
int iSystemFlag = 0;
dsICM.connect("ICMNLSDDB", "icmadmin", "password", "");
dsDef = (DKDatastoreDefICM) dsICM.datastoreDef();
dsAdmin = (DKDatastoreAdminICM) dsDef.datastoreAdmin();
dsCfg = (DKConfigurationMgmtICM) dsAdmin.configurationManagement();
lsCfg = (DKLSConfigurationMgmtICM) dsCfg.lsConfigurationMgmt();
lsDef = lsCfg.retrieveConfigParameters();
systemFlag = lsDef.getSystemFlag();
```

```

iSystemFlag = systemFlag | DKLSCfgDefICM.USE_VIEWID_LINK_AND_DELETE_DISABLED;
systemFlag = (short)iSystemFlag;
lsDef.setSystemFlag(systemFlag); // disable view acl for link and delete
lsCfg.defineConfigParameters(lsDef); // update library server with information
dsICM.disconnect();
dsICM.destroy();
DKLSCfgDefICM.java
class DKLSCfgDefICM
/**
 * Returns Use view for link and delete enabled state
 * @return true if the use view for link and delete is enabled,
 * false otherwise
 */
public boolean isUseViewForLinkAndDeleteEnabled();
// System Flag
public static final int USE_VIEWID_LINK_AND_DELETE_DISABLED = 16384;

```

### Example: C++

```

DKDatastoreDefICM* dsDef = NULL;
DKDatastoreAdminICM* dsAdmin = NULL;
DKConfigurationMgmtICM* dsCfg = NULL;
DKLSCfgDefICM* lsDef = NULL;
short systemFlag = 0;
long iSystemFlag = 0;
DKDatastoreICM* dsICM = new DKDatastoreICM();
dsICM->connect("ICMNLSDb","icadmin","password","");
dsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
dsAdmin = (DKDatastoreAdminICM*) dsDef->datastoreAdmin();
dsCfg = (DKConfigurationMgmtICM*) dsAdmin->configurationManagement();
lsCfg = (DKLSCfgDefICM*) dsCfg->lsConfigurationMgmt();
lsDef = lsCfg->retrieveConfigParameters();
systemFlag = lsDef->getSystemFlag();
iSystemFlag = systemFlag | USE_VIEWID_LINK_AND_DELETE_DISABLED;
systemFlag = (short)iSystemFlag;
lsDef->setSystemFlag(systemFlag); // disable view acl for link and delete
lsCfg->defineConfigParameters(lsDef); // update library server with information
dsICM->disconnect();
delete dsICM;
DKLSCfgDefICM.hpp
// System Flag
#define USE_VIEWID_LINK_AND_DELETE_DISABLED (short)16384
class DKLSCfgDefICM
/**
 * Returns Use view for link and delete enabled state
 * @return true if the use view for link and delete is enabled,
 * false otherwise
 */
DKEXPORT DKBoolean isUseViewForLinkAndDeleteEnabled();

```

---

## Implementing single sign-on in your applications

You can use single sign-on with Web or desktop applications in a WebSphere® Application Server environment.

Following is the authentication flow path, through the APIs, that you follow to implement this in your application:

1. A user enters their User ID and password through a browser application, and the request is sent to the WebSphere Application Server.
2. WebSphere Application Server authenticates the user by verifying the user information with an LDAP server.

3. If the user information is valid, WebSphere Application Server generates a cookie named LTPACookie. That cookie contains a Lightweight Third Party Authentication (LTPA) token.
4. The application logs on to DB2 Information Integrator for Content by calling `connectWithCredential()` in `DKDatastoreICM`, and using the LTPA cookie.  
Example:  

```
DKAuthenticationData credential = new DKAuthenticationData();
credential.setCredential(sLTPAToken);
//sLTPAToken can be a string representing the LTPA token generated

DKDatastoreICM dsICM = new DKDatastoreICM();

if (credential !=null){
 dsICM.connectWithCredential("ICMNLSDDB", credential, "");
} else{
 dsICM.connect("ICMNLSDDB", UserId, PWD, "") // normal connection with ID and PWD
}
```
5. The DB2 Information Integrator for Content API makes a call to WebSphere Application Server to validate the LTPA token.
6. WebSphere Application Server validates the LTPA token. If valid, a credential is created, otherwise an exception is thrown.
7. If a credential is created, it is wrapped into `DKAuthenticationData`, and the WebSphere Application Server user ID is extracted for use when logging on to DB2 Content Manager.

The following figure illustrates the authentication flow path through the APIs described above:

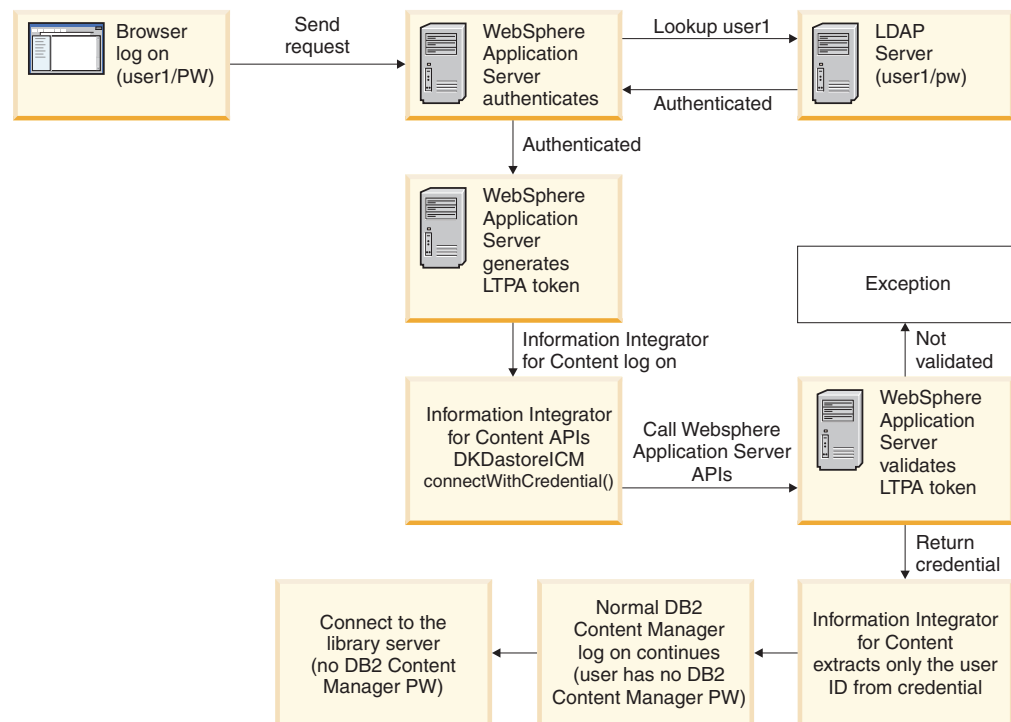


Figure 10. Single sign-on authentication flow through the APIs in a WebSphere Application Server environment

You can also use the `CMBConnection` bean to connect using single-sign. Following is an example using the `CMBConnection` bean:



```

public static void connect(CMBConnection connection, String dstype,
String server, String ltpaToken) throws Exception {

 // If the server name is followed by a parenthesized string,
 // use that string for the connect string.
 if (server.indexOf("(") > 0) {
 String connectString = server.substring(server.indexOf("(") + 1);
 server = server.substring(0, server.indexOf("("));
 if (connectString.endsWith("(")) {
 connectString = connectString.substring(0, connectString.length() - 1);
 }
 connection.setConnectString(connectString);
 }

 // Set properties on connection bean
 connection.setDsType(dstype);

 // Use the connect method to connect. This will
 // create the correct type of DKDatastore object (see Java API)
 // and call its connect method.
 System.out.println("Connecting to server");
 connection.connectWithCredential(server, ltpaToken, "");
 System.out.println("OK, Connected");
 // Enable display names support. This will cause CMBSchemaManagement,
 // CMBEntity, CMBAttribute, CMBItem to use display names when working
 // with DB2 Content Manager 8 (default is to use non-display names).
 connection.setDisplayNamesEnabled(true);

}

```

To use single sign-on in the manner described in the previous section, you must configure your system to allow trusted log on by completing the following steps:

1. Set the **Allow trusted logon** flag in the library server configuration. Using the system administration client, go to **Library Server Parameters** → **Configurations** → **Library Server Configuration**. Ensure that **Allow trusted logon** is selected.
2. Ensure that the database connection ID has the **UserDB2TrustedConnect privilege** set. To verify, using the system administration client, go to **Tools** → **Manage Database Connection ID** → **Change Shared Database Connection ID**. Ensure that **Password is required for all users** is not selected.
3. If you have not already done so, import your users into DB2 Content Manager from LDAP since WebSphere Application Server authenticates the user with an LDAP server.
4. Ensure that the privilege set for all DB2 Content Manager users contains the **AllowTrustedLogon** privilege. To verify, go to the system administration client and open the user's properties panel. Ensure that the privilege set for the user contains the **AllowTrustedLogon** privilege. For more information about privileges, see the administering information in the information center.

---

## Library server user exits for DB2 Content Manager

The users can customize the behavior of the library server by using the library server user exits.

### Important:

- For non-zLinux system, all C user exits must be designed by using 32 bits.
- For Linux on system z, all user exits must be designed by using 64 bits, except the ICMFetchContentExit, which still must be designed with 32 bits.

- ICMACLPriExit is a UDF and can be written as 32 or 64 bits. The 64-bit UDF is supported when operating under db2 64-bit instance.

The user exit sample directory has the bldUX.bat script for Windows and bldUX script for Unix, which are the samples of compilation and link parameters which can be used to build 32 or 64 bit user exits.

## User exit ICMFetchContentExit for changing text

This C user exit provides the option of changing text that is retrieved from the resource manager before it is passed to the service for full-text indexing. Conversion might include decryption, adding tags to create a structured document, or selection of only a subset of the object for indexing. This user exit is supported on DB2 platform only. Since Fix pack 2, DB2 Content Manager enhanced text search by using ICMFetchContentExit with a different interface.

### ICMFetchContentExit (old)

The object retrieved for full-text indexing is provided as input. This exit can transform the object in any way required, retrieving the new content in the input field if the length was not increased.

### Purpose

If the output length is unknown at the start of the processing, or the length might be increased, the exit code should allocate the necessary space using the address of the malloc() function that is passed. This is required to avoid problems when the DB2 Content Manager code is built with one compiler but the exit is built with a different compiler.

**Important:** If the library server attempts to load the user exit and the exit is not found, the exits continue normal operation, and does not provide an error when exiting out.

If the exit exists, it is automatically loaded. However, for performance reasons, the exit should only be loaded once, so run a db2stop/start command before attempting to load the exit.

You can customize the user exit in the following ways:

- The user exit function allocates memory for the decrypted object before sending it back to the text search user-defined function. In this case, the library server receives the object in the char \*\* pszDeObj buffer.
- The user exit function uses inplace decryption, which is where the decrypted object is contained in the same buffer as the input object. In this case, the library server receives the object in the original pszEnObj buffer.

### Parameters

The user exit is shown as:

```
extern C ICMFetchContentExit(long *psTIEFLAG, void *pvInputObject,
long *plInputObjectLength,void *ppvOutputObject,
long *plOutputObjectLength, void *malloc, long *plRC)
```

#### Input parameters

**long \*psTIEFLAG**

Pointer to TIEFLAG value passed from Client

```

char *pvInputObject
 Pointer to encrypted object sent to the exit

long * plInputObjectLength
 Length of encrypted object

char **ppvOutputObject
 Pointer to pointer to decrypted object set by the exit

long * plOutputObjectLength
 Length of decrypted object

void * mallocf
 Memory allocation function passed to the user exit

```

#### Output parameters

```

long *plRC
 Pointer to return code

```

#### Sample codes

The sample user exit routine and associated .mak files are located in the samples/server/exit directory of the product CD-ROM. For the sample user exit, the following files are installed under *IBMCMROOT*\samples\server\exit directory:

Table 16. File locations

| Sample name   | Purpose                                |
|---------------|----------------------------------------|
| icmdecxt.c    | C source code for user exit            |
| icmdecxt.def  | Standard DEF file for Windows platform |
| icmldecxt.mak | Make file for Linux platform           |
| icmndecxt.mak | Make file for Windows platform         |
| icmsdecxt.mak | Make file for Sun platform             |
| icmdecxt.mak  | Make file for AIX platform             |

The library must be copied to *ICMDLL*\databasename\DLL, where *ICMDLL* is a column PATHICMDLL in ICMSTSYSCONTROL table and *databasename* is the library server database name.

Source: icmdecxt.c

#### Return codes

- 0** Object was converted inplace or not converted. The input field should be returned for full text indexing.
- 1** Object was converted. Return the data in the output field for full text indexing, then free the memory.
- 2** No conversion is necessary

Any other return codes results in an error in conversion. If plOutputObjectLength is not zero, ppvOutputObject contains message text to write to DB2 Content Manager server log.

## ICMFetchContentExit (new)

Since Fix Pack 2, DB2 Content Manager enhances the text search function by using the function ICMFetchContentExit.

DB2 Content Manager:

- Enables indexing of metadata or content by providing a mechanism in ICMFetchFilter
- Supports a different user exit routine for each text index

The function ICMFetchContentExit is already available in the DLL icmdecxt (on UNIX®) or icmdecxt.dll (on Windows®). The new function is the same function with a different interface. The new DLLs are icmxlsfcn on UNIX systems and icmxlsfcn.dll on Windows. The last character, n, is derived from the value of the TIEFLAG parameter.

Unlike the existing user exit routine, which is called only after retrieving the object from the resource manager, the new user exit routine is called before object retrieval. Depending on what is returned, the new user exit routine is called again after retrieval.

Use of the existing ICMFetchContentExit function (from the icmdecxt DLL) is not impaired by the new implementation. The load precedence is: If no user exit routine is found, none is loaded. If the new user exit routine is found, based on the value of the TIEFLAG parameter, then the new user exit routine (from the icmxlsfcn DLL) is loaded. Otherwise, the existing user exit routine (from the icmdecxt DLL) is loaded. The new user exit routine uses the same log file that ICMFetchFilter does.

The user exit, ICMFetchContentExit (ICMFetchContentParms1 \*IFP) enhances the text search.

```
/* ----- */
/* ICMStFetchContentParms1 data structure */
/* ----- */

typedef enum
{
 /* Return codes for user exit when called BEFORE retrieving */
 /* the object from the resource manager */

 ICMRcContinueRetrieving = 0, /* No text returned - continue with */
 ICMRcTextReturned = 1, /* retrieve from the resource manager */
 ICMRcTextReturnedSkipInso = 4, /* Text returned - call INSO to convert */
 /* Object will not be retrieved from RM */
 /* Text returned - do not call INSO */
 /* Object will not be retrieved from RM */

 /* Return codes for user exit when called AFTER retrieving */
 /* the object from the resource manager */

 ICMRcInplaceConversion = 0, /* Object was not converted or was */
 /* converted in place. Continue processing */
 /* by calling INSO if the MIMETYPE is for */
 /* a text object. */
 ICMRcOutputGenerated = 1, /* Object was converted and returned */
 /* in memory allocated by the function */
 /* pMalloc. INSO conversion should not */
 /* be called if MIMETYPE indicates text. */

 ICMRcConversionError=2, /* Error in conversion; No text returned */
 /* to NSE; output object written to UDF log*/
}
```

```

 ICMRcInplaceConversionSkipInso=3, /* Same as 0 but skip INSO conversion */
 ICMRcOutputGeneratedSkipInso=4, /* Same as 1 but skip INSO conversion */

 ICMRcWarning = -1 /* The object cannot be converted, and */
 /* a "warning" should be returned to NSE */
 /* to skip this document */
 } ICMEnRc;

typedef struct ICMStFetchContentParmsV1
{
 /******
 /* The following parameters are passed into the exit and should not
 /* be changed.
 /******
 size_t uSizeOfStruct; /* Size of the structure */
 short sExitVersion; /* Used for consistency checking */
 short sExitType; /* Version number of the exit. The
 /* structure shown here is for V1.
 /* 1: Pre-retrieve (called before
 /* object is retrieved from RM)
 /* 2: Post-retrieve (called after
 /* object was retrieved from RM)
 long lComponentTypeID; /* Not currently supported */
 char szContentType[ICM_SIZE_CONTENT_TYPE]; /* MIME type for post-retrieve */
 short sTIEFlag; /* TIEFLAG value can be used to
 /* control processing in the exit
 char szItemId[ICM_SIZE_ITEM_ID]; /* ITEMID being processed
 short sItemVersionId; /* VersionID being processed
 void *pvInputObject; /* Pointer to the object to be
 /* processed. The object may be
 /* changed by the exit if RC=0 or 2
 size_t uInputObjectSize; /* Size of the input object in bytes
 void * pfMalloc; /* This pointer to malloc must be used
 /* to allocate memory returned in
 /* pvOutputObject
 FILE * pstDebugFile; /* If not null, pointer to a file
 /* that may be used for debugging via
 /* fprintf, fflush. DO NOT CLOSE.

 /******
 /* The following parameters are set by the exit
 /******

 void * pvOutputObject; /* Pointer to the output object when
 /* RC=1 or 2
 size_t uOutputObjectSize; /* Length of the object in bytes

 char szErrorMessageText[ICM_SIZE_MESSAGE_TEXT]; /* Error message
 ICMEnRc ReturnCode; /* Return code

} ICMStFetchContentParmsV1;

```

## User exit ICMLogonExit for logon

This C user exit is called at logon time and it provides an interface to validate a userID, password, user counting, and LDAP authentication.

### Purpose

The external name for this user exit is ICMXLSLG.DLL. Place the DLL in the directory: ICMROOT/database name/DLL to use this user exit. This user exit is supported on DB2 and z/OS.

## Parameters

```
extern C ICMLogonExit(pszLanguage, pszUserID, pszPassword,
pszNewPassword, pszApplication, psUserDomain, pszLDAPInfo,
pszDatabaseName, pszSchemaName, plRC, plReason, plExtRC, plExtReason)
```

### Input parameters

The following are the input parameters:

**char \*pszLanguage**  
Language code

**char \*pszUserID**  
User ID

**char \*pszPassword**  
Decrypted password

**char \*pszNewPassword**  
New decrypted password (if new password was provided)

**char \*pszApplication**  
Name of the application

**short \*psUserDomain**  
User domain

**char \*pszLDAPInfo**  
The path in the LDAP server for this userID

**char \*pszDatabaseName**  
The database name of the library server

**char \*pszSchemaName**  
The database schema name of the library server

### Output parameters

The following are the output parameters:

**long \*plRC**  
Return code

**long \*plReason**  
reason code

**long \*plExtRC**  
Use this to return any extra application-specific error information

**long \*plExtReason**  
Use this to return any extra application-specific error information

Set reason code (plReason ) according to the following rules:

**\*plReason = 0**  
ICMLogon will do password validation

**\*plReason = 1**  
ICMLogon will bypass password validation.

## Sample codes

A code sample is provided in the ICM root directory in: /samples/icmxslsg.c. For EIP, the code sample is in the EIP root directory in: /samples/icmxslsg.c.

For the sample user exit, the following files are installed under *IBMCMROOT\samples\server\exit* directory:

*Table 17. File locations*

| Sample name  | Purpose                                |
|--------------|----------------------------------------|
| icmxslsg.c   | C source code for user exit            |
| icmxslsg.def | Standard DEF file for Windows platform |
| icmnlsg.mak  | Make file for Windows(R) platform      |
| icmslsg.mak  | Make file for Sun platform             |
| icmxslsg.mak | Make file for AIX platform             |

The library must be copied to *ICMDLL\databasename\DLL*, where *ICMDLL* is a column *PATHICMDLL* in *ICMSTSYSCONTROL* table and *databasename* is the library server database name.

## Return codes

The following list contains the return codes (\*plRC) expected when logging on to DB2 Content Manager:

### 0 - RC\_OK

Validation passed. ICMLogon continues normal execution.

### 7123 - RC\_INVALID\_PARAMETER

The name of any invalid pointer or value is logged. Logon is denied.

### 7015 - RC\_UNEXPECTED\_SQL\_ERROR

SQL error. Logon is denied.

### 7172 - RC\_INVALID\_PASSWORD

The password does not match the password defined for this user. Logon is denied.

### 7173 - RC\_MAX\_LOGON\_PASSWORD\_RETRY

The maximum number of attempts with wrong password has been reached. Logon is denied.

### 7098 - RC\_INVALID\_NEWPASSWORD

Set only by the exit should the new password not match the costumer's rules. Logon is denied.

### 7171 - RC\_PASSWORD\_EXPIRED

The password must be changed. Call ICMLogon again with a new password. Logon is denied.

### 7160 - RC\_LOGON\_MAX\_USER\_ERROR

The maximum number of concurrent users has been reached. Logon is denied.

### 7094 - RC\_ALREADY\_LOGGED\_ON

The UserID is already logged on to the Content Manager system. Logon is denied.

### 7006 - RC\_DLL\_LOAD\_ERROR

A DLL could not be loaded. Logon is denied.

### 7011 - RC\_GET\_PROC\_ADDRESS\_ERROR

A procedure could not be obtained. Logon is denied.

#### 4751 - RC\_LOGON\_MAX\_USER\_WARNING

The maximum number of concurrent users has been reached. Logon is allowed, but a warning message should be displayed.

##### Related information

Enhancements to text search

## User exit myExit for document routing

This user exit is called after creating a work package (using the start process), when a work package is moved off a work node, to move to a work node, or when the overload limit is reached.

### Purpose

At the time the exit is called, the work package exists in the work package table, and can be retrieved using the componentID as the identifier of the row in table ICMUT00204001.

### Parameters

```
typedef struct ICMCONTAINERDATA_STRUCT
{
 char szContainerName[33]; Container Data variable name
 char szContainerVal[255]; Container Data variable value
} ICMCONTAINERDATA_STRUCT;
typedef struct _ICMUSERSTRUCT
{
 long lUserEvent;
 char szWPCompID[19];
 char szWPItemID[27];
 short sWPVersionID;
 char szRouteSel[33];
 short sUpdateFlag;
 short sNumContainerData;
 pContainerDataStructIn
 pContainerDataStructOut
 const ICMCONTAINERDATA_STRUCT *pContainerDataStructIn;
 ICMCONTAINERDATA_STRUCT *pContainerDataStructOut;
 void * mallocf;
} ICMUSERSTRUCT;
```

#### Input parameters

##### lUserEvent

Indicates in which Work Node Event this user exit was called. Values:

- 1        The work package is entering the work node.
- 2        The work package is leaving the work node.
- 3        The work package is at a work node, exceeding the overload limit at this work node.

##### szWPCompID

The work package component ID passed to this user exit

##### szWPItemID

The work package item ID passed to this user exit.

##### sWPVersionID

The work package Version ID passed to this user exit

##### sNumContainerData

Number of ICM container data structures that is sent to this exit in pContainerDataStructIn



### **pContainerDataStructIn**

Contains container data sent to the user exit

### **Output parameters**

#### **szRouteSel**

The route the work package takes on return from this user exit

#### **sUpdateFlag**

1 User exit is returning updated container data

0 User exit is not returning updated container data

#### **sNumContainerData**

Number of ICM container data structures that is returned on pContainerDataStructOut

### **pContainerDataStructOut**

Contains container data returned by the user exit

#### **mallocf**

Pointer to the C malloc function that this user exit needs to use to allocate memory for the output container data structure will be returned in pContainerDataStructOut

## **Sample codes**

A sample code is provided in *IBMCMROOT\samples\server\exit*. For the sample user exit, the following files are installed under *IBMCMROOT\samples\server\exit* directory:

Table 18. File locations

| Sample name   | Purpose                                                                                 |
|---------------|-----------------------------------------------------------------------------------------|
| icmdruext.c   | C source code for user exit                                                             |
| icmdruext.def | Standard DEF file for Windows platform                                                  |
| wxv2tue.h     | Header file for user exit constants, prototypes                                         |
| bldexit.bat   | Compile DB2 Content Manager library server user exit using Microsoft Visual Studio 2003 |

The library must be copied to *ICMDLL\databasename\DLL*, where *ICMDLL* is a column PATHICMDLL in ICMSTSYSCONTROL table and *databasename* is the library server database name.

## **User exit ICMValidatePassword for validating passwords**

The password validation user exit is called to validate whether the new password meets the required guidelines.

### **Purpose**

You can customize your user exit to meet your password guidelines. This user exit is supported on DB2 and z/OS.

The library server allows users to enforce password syntax validation rules by using the user exit ICMPLSVP library. The ICMPLSVP library user exit is called when:

- The users are creating their profile or updating the user password

- The users are changing their DB2 Content Manager login password

All the declarations and definitions are in the header file icmuxcom.h.

For this user exit, the library server initializes once when either the user profile is created or the user password is changed. If the library server finds the ICMPLSVP library at the specified location, it loads the user exit and runs the ICMValidatePassword function. If the library server does not find the ICMPLSVP library at the specified location, the library server does not check for the user exit again until after the database is stopped or started.

Therefore, to deploy the user exit, you must stop or start the database server for the library server to use the new user exit.

After the user exit is called, it returns either UX\_HOLD or UX\_RELEASE. If the user exit returns UX\_HOLD, the library server does not unload the user exit and keeps it in memory to enhance performance. The return value, UX\_RELEASE causes the library server to release the user exit and attempts to reload again on the next invocation. In this case, the user exit is reloaded every time.

**Important:** If the password user exit is placed under cmgmt/ls/icmnlbdb/DLL, the user exit is used by the library server to modify the password for DB2 Content Manager users. However, if you are trying to modify password of DB2 Content Manager administration user ( icmadmin) from DB2 Content Manager logon window or pClient and eClient logon window, the user exit is not applied because the API calls DB2 database directly instead of the library server to change the password.

## Parameters

The function that is invoked in the user exit ICMValidatePassword is declared extern C as shown:

```
int ICMValidatePassword (ICM_UX_PASSWORD_VALIDATION_STRUCT *pInputParams,
 ICM_UX_DIAGNOSTIC_RETURN_STRUCT *pReturnParams);
```

### Output parameters

Common user exit output parameters, return codes and diagnostic string.

```
typedef struct ICM_UX_DIAGNOSTIC_RETURN_STRUCT {
 int returnCode; //output: RC_OK, RC_ERROR
 int returnReasonCode; //output: user-defined return codes
 char diagString[2048+1]; //output. additional text info
} ICM_UX_DIAGNOSTIC_RETURN_STRUCT;
```

#### returnCode

Return code. The possible return codes from the ICMValidatePassword user exit are:

##### RC\_OK(value 0)

Password validation succeeded.

##### RC\_ERROR(value -1):

Password validation failed.

#### returnReasonCode

User-defined reason code. The returnReasonCode indicates the reason why the password failed to validate. This reason code is returned to the client or the application.

### **diagString**

User-defined error or warning message is logged in the library server log if it is not null.

### **Input parameters**

Input parameters for the password validation user exit.

```
typedef struct _ICM_UX_PASSWORD_VALIDATION_STRUCT {
 char currentLSVersion[128+1];
 char userId[32+1];
 char newPassword[48+1];
} ICM_UX_PASSWORD_VALIDATION_STRUCT;
```

### **currentLSVersion**

Level of the library server. This parameter might be used to check for compatibility of the user exit and the library server version.

### **userId**

DB2 Content Manager user ID.

### **newPassword**

New password.

### **Return values**

The user exit function returns:

#### **UX\_HOLD (value 0)**

The user exit library is not released after return.

#### **UX\_RELEASE (value 1)**

The user exit library is released after return.

### **Sample codes**

A sample code is provided in *IBMCMROOT\samples\server\exit\icmplsvp\**. For the sample user exit, the following files are installed under *IBMCMROOT\samples\server\exit* directory:

*Table 19. File locations*

| Sample name  | Purpose                                                                                                          |
|--------------|------------------------------------------------------------------------------------------------------------------|
| icmplsvp.c   | C source code for user exit                                                                                      |
| icmuxcom.h   | Common header file for user exit constants and prototypes                                                        |
| icmplsvp.def | Standard DEF file for Windows platform                                                                           |
| bldUX.bat    | Compile DB2 Content Manager library server user exit using Microsoft Visual Studio 2003                          |
| bldUX        | UNIX <sup>®</sup> script to compile DB2 Content Manager library server user exit on AIX, Sun and Linux platforms |

The library must be copied to *ICMDLL\databasename\DLL*, where *ICMDLL* is a column *PATHICMDLL* in *ICMSTSYSCONTROL* table and *databasename* is the library server database name.

### **Return codes**

The following list contains the library server return codes when you log in to DB2 Content Manager:

## 7109 - ICM\_PASSWORD\_VALIDATION\_FAILED

The new password is not valid when you are trying to create a user or change the password.

The following exception message is displayed:

```
DGL7280A: The new password failed to pass the validation rules.: : U001;
ICM0001: This is an unexpected internal error or an unrecognized error code.
Run the application again with server trace set to Detail and Data.
Save the server log and see your IBM service representative.
: 7109 (STATE) : [LS RC = 7109, LS reasonCode = 2]
```

## 7011 - RC\_GET\_PROC\_ADDRESS\_ERROR

The user exit does not contain the required function name. See the library server log for more information.

## ACL user exit routines

DB2 Content Manager provides two access control user exit routines that override the built-in access control mechanism: ICMACLPPrivExit and ICMGenPrivExit. The user must build both of them to complete the override.

### ICMACLPPrivExit

#### Purpose

Use this exit routine to determine whether the user has the authority to perform the requested function. This is not a C user exit routine, but a C database UDF. This UDF is used when DB2 Content Manager checks the privileges inside an SQL statement. This user exit is supported on DB2 and z/OS platforms.

#### Parameters

The following example shows the C function declaration for the ACL privilege exit routine UDF.

```
void SQL_API_FN ICMACLPPrivExit(
 char *pszUserID,
 char *pszApplicationID,
 char* APIAction,
 char *pszProcedureName,
 char *InfoType,
 char *ItemID,
 char *ViewID,
 long *plGenPrivCode,
 short * pszUserID _ind,
 short * pszApplicationID _ind,
 short * APIAction _ind,
 short * pszProcedureName _ind,
 short * InfoType _ind,
 short * ItemID _ind,
 short * ViewID _ind,
 short * plGenPrivCode _ind,
 char *sqlstate, /* SQL STATE */
 char *funcname, /* function name */
 char *specname, /* specific function name */
 char *msgtext /* message text insert */
)
```

#### Input parameters

##### pszUserID

User ID (maximum number of characters is 32)

**pszApplicationID**

Application ID (maximum number of characters is 32667 - reserved)

**pszHostname**

Host name of client (maximum number of characters is 128)

**APIAction**

The API actions are:

- 0 Retrieve
- 2 Update
- 3 Delete
- 4 Not available

**pszProcedureName**

Library server stored procedure name (20 characters)

**InfoType**

Set the following parameter for the information type related to:

- 0 Items
- 2 Views
- 3 Item types or item type views

**ItemID**

Item ID (26 characters)

**ViewID**

Item type ID, item type view ID, or view ID

**pIGenPrivCode**

General privilege required for the action

**Output parameters****pIRC**

Pointer to return code

**Sample codes**

A sample code is provided in *IBMCMROOT\samples\server\exit*. For the sample user exit, the following files are installed under *IBMCMROOT\samples\server\exit* directory:

Table 20. File locations

| Sample name   | Purpose                                                      |
|---------------|--------------------------------------------------------------|
| icmaclxt.c    | C source code for user exit                                  |
| icmaclxt.def  | Standard DEF file for Windows platform                       |
| icmaclxt.exp  | Standard EXP file for Windows platform                       |
| aclnvar.mak   | Global variables needed by the Windows(R) User Exit makefile |
| icmlaclxt.mak | Make file for Linux(R) platform                              |
| icmnaclxt.mak | Make file for Windows(R) platform                            |
| icmsaclxt.mak | Make file for Sun platform                                   |
| icmxaclxt.mak | Make file for AIX(R) platform                                |

The library must be copied to <sqlib>\function, where <sqlib> is the database instance home directory for DB2.

### Return codes

0        RC\_OK validation that the ICMACLPPrivExit routine grants the ACL to query

### Non-zero

Declines access to the ACL query

## ICMGenPrivExit

### Purpose

Use this exit routine to determine whether the user has the general privilege to perform the requested function on a particular item or view. This user exit is supported on DB2 and z/OS.

### Parameters

The following example shows the ICMGenPrivExit user exit routine. The parameters provide information that you can use in your security access logic.

```
extern long ICMGenPrivExit(
 char *pszUserID,
 char *pszApplicationID,
 long *pIgenPrivCode,
 char *pszProcedureName,
 long *pICMDDecision,
 long *pICMDDecReason,
 long *pIRC
)
```

### Input parameters

#### pszUserID

User ID (maximum number of characters is 32)

#### pszApplicationID

Application ID (maximum number of characters is 32667 - reserved)

#### pIgenPrivCode

General privilege required for the action

#### pszProcedureName

Library server stored procedure name (20 characters)

#### pICMDDecision

Library server decision

0        Grant access

#### Other codes

Look up library server error codes

#### pICMDDecReason

Extended return code if available

### Output parameters

#### pIRC

Pointer to return code

## Sample codes

A sample code is provided in *IBMCMROOT\samples\server\exit*. For the sample user exit, the following files are installed under *IBMCMROOT\samples\server\exit* directory:

Table 21. File locations

| Sample name   | Purpose                                |
|---------------|----------------------------------------|
| icmgenxt.c    | C source code for user exit            |
| icmgenxt.def  | Standard DEF file for Windows platform |
| icmgenxt.exp  | Standard EXP file for Windows platform |
| icmlgenxt.mak | Make file for Linux(R) platform        |
| icmngenxt.mak | Make file for Windows(R) platform      |
| icmsgenxt.mak | Make file for Sun platform             |
| icmxgenxt.mak | Make file for AIX(R) platform          |

The library must be copied to *ICMDLL\databasename\DLL*, where *ICMDLL* is a column *PATHICMDLL* in *ICMSTSYSCONTROL* table and *databasename* is the library server database name.

## Return codes

**0** RC\_OK validation that the ICMACLPPrivExit routine grants the ACL to query

**Non-zero**  
Declines access to the ACL query

---

## 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 connection 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 that is not a database user ID, you can make the following changes:

### For a federated database

In the cmbds.ini file, update the FEDSERVERREPTYPE value to DB2CON instead of DB2 for a particular federated database. It should look like this: FEDSERVERREPTYPE=DB2CON. When the FEDSERVERREPTYPE is set to DB2CON, the connector logs on using the database connection ID first, and the connector then passes 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 fails. The connector automatically disconnects from the database and reconnects using the federated user ID to log on to the database and then passes the federated database user ID to the server.

For the Java and C++ DKDatastoreFed connector connect method, there is another connect string option (REPTYPE=DB2CON) that you can specify so that a particular instance of the datastore has the behavior mentioned above.

#### **For a DB2 Content Manager Version 8 database**

In the cmbicmsrvs.ini file, 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

After you change the ICMSEVERREPTYPE value, the DB2 Content Manager connector logs on to the server using the database connection ID first, and then passes 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 fails. The connector automatically disconnects from the database and reconnects using the DB2 Content Manager Version 8 user ID to log on to the database, and then passes 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 (REPTYPE=DB2CON), so that a particular instance of the datastore has the behavior mentioned above.

---

## **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 communication between thin client or applet applications and the resource manager. Use SSL if your application accesses resource manager objects and runs outside of a secure boundary, such as a firewall.

### Requirements

- 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 application 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, edit the `cmbrm.ini` file, which you can find in the same directory as other properties files. The `cmbrmenv.properties` file contains an entry called `CMCFGDIR`, which points to the location of `cmbrm.ini` file. In the `cmbrm.ini` file, set the `RM_SSL_FOR_URL_RETRIEVES` setting to 1. If the setting does not exist, add it.

After you verify that the `RM_SSL_FOR_URL_RETRIEVES` setting is correct, your application can use the HTTPS URL to securely communicate with the resource manager.

Note that `RM_SSL_FOR_URL_RETRIEVES` is a global setting, and that after you set it to 1, all URL-based retrieves with all resource managers in the system will use SSL. That means that if your system is set up to work with multiple resource managers, all of 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 for the DB2 Content Manager Version 8 connector for objects stored on a fixed disk within a Windows, AIX, Solaris resource manager.

**Important:** 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 earlier levels, irrecoverable destroy is processed as a normal deletion.

To use irrecoverable destroy, you must specify the `DKConstant` option `DK_CM_DESTROY_DELETE` in operations that result in the deletion of an object. The following list summarizes 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 for deleting objects. The following list summarizes the APIs and associated methods you can use to 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[] rmsmpairs)
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 operation. 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 operations.

## **Asynchronous replication on 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. If objects were stored in DB2 Content Manager before the creation and association of replication rules, DB2 Content Manager provides a utility called IMPORTREPLICA for this purpose. For more information about the IMPORTREPLICA utility, see the administering information in the information center.

Asynchronous replication functionality is encapsulated in a standalone module of the z/OS resource manager. To complete an asynchronous replication process, complete the following steps:

1. Run ICMRMAP.JCL to start the asynchronous replication process, ICMRMAP.JCL. The input parameter list passed into ICMRMAP.JCL from ICMRMAP.JCL is then verified. The input parameter list is as follows:

**DB2SUBSYSID (DB2C)**

4 bytes: The **DB2SUBSYSID** indicates the location where the z/OS resource manager database is located.

**DB2 Content Manager library server database (NQADB2C)**

Alias used to locate a given database within the specified DB2 UDB subsystem.

**Library server location (LOCAL/REMOTE)**

This value indicates whether the library server database resides within the same DB2 UDB subsystem as that of the z/OS resource manager, or resides externally.

**Plan name (OSAPFVTA)**

8 bytes: The bind plan for the asynchronous replication process.

**User ID (IFVTA)**

The DB2 Content Manager user with system administrator authority.

**User Password (IFVTA1)**

The DB2 Content Manager system administration password.

**Resource Manager Name (IMWEBSR1)**

This name indicates to which z/OS resource manager the asynchronous process is associated.

**Sleep value (300)**

The duration (in seconds) for the process to pause before reprocessing items marked for deletion.

**Trace level (0 – 3)**

Log level. The current supported values are 0, 1, 2, and 3.

**Level 0: ERROR**

Logs only the asynchronous replicate summary report and errors.

**Level 1: INFO**

Logs the function entry and exit status plus ERROR.

**Level 2: DETAIL**

Logs function input and output, success or failure results, and INFO.

**Level 3: DEBUG**

Logs verbose statements used for debugging and INFO.

2. Populate the `ReplicaItemStruct` struct. If all of the input parameters are valid, then the asynchronous replication process proceeds to connect to the library server database to retrieve the data-collection and resource manager 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 replicate in the resource manager `Asynch` table and store them in the `ReplicaItemStruct` struct. This is the struct that drives the asynchronous replication process.
3. After the `ReplicaItemStruct` struct is populated, asynchronous replication calls several helper functions to complete its tasks. The most important functions are:

**mapRMDData()**

This function takes the `ReplicaItemStruct` struct as one of its parameters and uses the information in it to validate the resource managers listed in the `Asynch` table against the resource manager information retrieved

from the library server to ensure that they exist. If the verification is successful, then the source and target resource managers are mapped appropriately in the `ReplicaItemStruct` struct.

#### **mapCollData()**

This function also takes the `ReplicaItemStruct` struct as one of its parameters and uses the information in it to validate the collections listed in the `Asynch` table against the collection information retrieved from the library server to ensure that they exist. If the verification is successful, the source and target collections are mapped appropriately in the `ReplicaItemStruct` struct. This function is called only after the source and target resource managers are mapped successfully.

#### **checkOutItems()**

This function also takes the `ReplicaItemStruct` struct as one of its parameters and uses the information in it to check out the items to be replicated. When an item is checked out, it is locked and its information is also written into the `CHECK OUT` table in the library server.

#### **sendReplicasToTargetRM()**

This function takes the `ReplicaItemStruct` struct as one of its parameters and uses the information in it to determine which item needs to be replicated and from which resource manager and collection to which resource manager and collection.

`sendReplicasToTargetRM()` accomplishes its tasks in several steps:

- a. First, it checks with OAM to determine if the object exists or not. If the object exists, then it attempts to retrieve the object.
- b. Second, it determines which target resource manager collection this object needs to be sent to.
- c. Finally, an HTTP request with the target resource manager collection information and the object is sent to the target resource manager collection. Hence, the object is replicated.

## **Table definitions and column descriptions**

*Table 22. ICMRMCONTROL*

| ICMRMCONTROL                                                                                                                                                                                                                                                                                                                                                                                                                                                 |          |     |      |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-----|------|
| STOPREPLICATOR                                                                                                                                                                                                                                                                                                                                                                                                                                               | SMALLINT | NOT | NULL |
| STOPREPLICATOR: (0/1) The REPLICATOR bit determines if the ICMMRMAP 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 the bit is set to one, then it will stop after one round of processing. If it is 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 23. 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

The ItemId value that is generated by the library server during the object store transaction. Example: A1001001A03L09B64813I92553

## VERSIONID

In the case where versioning is enabled, this value denotes a given instance of an object. Example: 1

## OBJID

The OBJID is the item ID and version ID combined and delimited with periods (.), represents a unique DB2 Content Manager object locator within Object Access Method (OAM). Example:  
A1001001.A03L09.B64813.I92553.V001

## COLLNAME

The sourcecollname is the collection under which the object was originally stored and is the source of the prefetch operation. Example: CLLCT001

Table 24. ICMSTITEMSTODELETE

| ICMSTITEMSTODELETE |           |          |
|--------------------|-----------|----------|
| ITEMID             | CHAR (26) | NOT NULL |
| VERSIONID          | SMALLINT  |          |
| RCODE              | SMALLINT  |          |

## RCODE

The name of the resource manager, given by the user, is used to map to RCODE, which in turn points to an entry in the table. Example: 1

## Storing content in a user-specified collection by using Java or C++ API

When you store objects, you can specify an alternate resource manager collection in your custom application by changing the DKLobICM class.

The ItemSetRMCollection privilege is added to the existing set of privileges in the DKLobICM class. The ItemSetRMCollection privilege allows you to override the resource manager collection that is set by the administrator. The resource manager name and collection name must be valid and must exist in the DB2 Content Manager system. If you do not specify the resource manager name and the collection name, you might get a DKUsageError exception. You must also define the collection name in the specified resource manager and not in other resource managers. If you do not define the collection name in the specified resource manager, you might get a DKUsageException exception. For more information, see DKLobICM class in *Application Programming Reference*.

## Example

To store the content associated with document part and resource items with any user-specified location at an item level, the Java and C++ API provides the following method:

**DKStorageManageInfoICM::setStoreLocation( string ResourceManagerName, string collectionName)**

Ensure that the parameters of this method, the resource manager name and the collection name are valid.

### Resource item

```
DKTextICM xdo = (DKTextICM) datastore.createDDO(itemTypeName,
(int) DKConstantICM. DK_ICM_XDO_TEXT_CLASS_ID);
DKPidICM apid = (DKPidICM)xdo.getPidObject();
apid.setVersionNumber("1"); xdo.setMimeType(mimeType);
DKStorageManageInfoICM storageInfo = (DKStorageManageInfoICM)
xdo.getExtension("DKStorageManageInfoICM");
storageInfo.setStoreLocation(resourceManagerName,
collectionName);
xdo.add(inputFileName); // Become persistent
```

### Document part

```
DKDDO ddo_doc = datastore.createDDO(itemTypeName,
DKConstant.DK_CM_FOLDER);
DKParts partColl = null;
short dataId =
ddo_doc.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
DKConstant.DK_CM_DKPARTS);
if (dataId == 0) {
dataId = ddo_doc.addData(
DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS);
partColl = new DKParts();
ddo_doc.setData(dataId, partColl);
} else {
partColl = (DKParts)ddo_doc.getData(dataId);
if (partColl == null){
partColl = new DKParts();
ddo_doc.setData(dataId, partColl);
}
}
DKTextICM part_xdo = (DKTextICM)datastore.
createDDO(partName,
(int) DKConstantICM. DK_ICM_XDO_TEXT_CLASS_ID);
DKPidICM apid = (DKPidICM)part_xdo.getPidObject();
apid.setVersionNumber("1");
part_xdo.setMimeType(mimeType);
DKStorageManageInfoICM storageInfo =
(DKStorageManageInfoICM)
part_xdo.getExtension("DKStorageManageInfoICM");
storageInfo.setStoreLocation(resourceManagerName,
collectionName);
part_xdo.loadDataFromFile(inputFileName);
part_xdo.setAddLocationOption("MEMORY");
partColl.addElement(part_xdo);
ddo_doc.add();
```

In the previous example, you can substitute the `DKStorageManageInfoICM::setStoreLocation( string resourceManagerName, string collectionName)` method by the `DKStorageManageInfoICM::setStoreSite(string resourceManagerName) method and the DKStorageManageInfoICM::setCollectionName(string collectionName) method.`

**Important:** This feature is for storing only newly created items in the resource manager. For update functions, if you specify the **RMName** and **SMSCollectionName** configurable parameters, DB2 Content Manager does not consider your input and continues updating the **RMName** and **SMSCollectionName** parameter pair where the object was stored.

## API for resource manager in Version 8.4

The API in DB2 Content Manager Version 8.4 enables the customers to remotely manage the resource manager by APIs, system administration client, and WebSphere Application Server.



DB2 Content Manager Version 8.4 provides an interface for the applications to:

- Configure Tivoli® Storage Manager (TSM) properties from a remote location and store the properties in the resource manager database
- Control object migration based on high and low threshold values
- Stop or start the resource manager and check the status of various resource manager services

The API enhancements of the resource manager eliminates the need of the system administrators to have physical access to the machine hosting the resource manager. In DB2 Content Manager Version 8.4, the resource manager also supports WebSphere Application Server network clusters. The ability to manage the clusters in a network provides the ability to manage the resource manager from a remote location.

**Important:** The enhancements of the resource manager are enabled by the ICM connector Java APIs. You must install DB2 Content Manager Version 8.4 to use the API enhancements of the resource manager.

The following classes in the package `com.ibm.mm.sdk.rm` are modified to support these interfaces.

## DKRMConstantSMS

The following constants are added to the `DKRMConstantSMS` class to support flexible migration policies based on high and low threshold.

Table 25. New constants in `DKRMConstantSMS` class

| Constants                                   | Purpose                                                                                        | Value |
|---------------------------------------------|------------------------------------------------------------------------------------------------|-------|
| <code>SMS_ALLOWED_MIN_LOW_THRESHOLD</code>  | Indicates the minimum allowed value for a low threshold (resume threshold for enabling volume) | 0     |
| <code>SMS_ALLOWED_MAX_HIGH_THRESHOLD</code> | Indicates the maximum allowed value for a high threshold (threshold for disabling volume)      | 100   |
| <code>SMS_ALLOWED_MIN_HIGH_THRESHOLD</code> | Indicates the minimum allowed value for a high threshold (threshold for disabling volume)      | 1     |

## DKRMServerDefinitionDefSMS

The following methods are added to the `DKRMServerDefinitionDefSMS` class.

Table 26. New methods in `DKRMServerDefinitionDefSMS` class

| Methods                          | Purpose                                                                                                                                                                                            |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>setDSMConfig</code>        | Sets the fully qualified path for the TSM client options file                                                                                                                                      |
| <code>getDSMConfig</code>        | Returns the fully qualified path for the TSM client options file                                                                                                                                   |
| <code>setTSMBufferSize</code>    | Sets the buffer size to use for data transfer between the resource manager and the TSM server                                                                                                      |
| <code>getTSMBufferSize</code>    | Returns the buffer size to use for data transfer between the resource manager and the TSM server                                                                                                   |
| <code>setTSMFileSpaceName</code> | Sets the TSM file space name. TSM determines the buffer size of the content location that must be used for data transfer with the TSM server by using the <code>setTSMFileSpaceName</code> method. |



Table 26. New methods in DKRMServerDefinitionDefSMS class (continued)

| Methods             | Purpose                                                                                                                                                                                  |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| getTSMFileSpaceName | Returns the TSM file space name. TSM determines the buffer size of the content location that must be used for data transfer with the TSM server by using the getTSMFileSpaceName method. |
| setTSMFileSpaceInfo | Sets an identifier of the resource manager objects within the TSM file space                                                                                                             |
| getTSMFileSpaceInfo | Returns an identifier of the resource manager objects within the TSM file space                                                                                                          |

See the Javadoc reference documentation for more information.

## Example

The following example demonstrates the use of some of the above methods in the DKRMServerDefinitionDefSMS class:

```
DKRMConfigurationMgmtICM _rmCfgMgmt =
new DKRMConfigurationMgmtICM(dsICM);
DKRMServerDefinitionDefSMS _serverdefObj = null;
DKResourceMgrDefICM _resMgrDef =
 (DKResourceMgrDefICM)_rmCfgMgmt.retrieveResourceMgr(rmName);
// Connect to resource manager
DKRMStoreSMS _rmStoreSMS = new DKRMStoreSMS(_resMgrDef);
_rmStoreSMS.connect(_resMgrDef, "");
_serverdefObj = new DKRMServerDefinitionDefSMS(_rmStoreSMS);
_serverdefObj.retrieve();
// Set fully qualified path for TSM client properties file
_serverdefObj.setDSMConfig("/opt/tivoli/tsm/client/api/bin/dsm.opt");
// Sets the Tivoli Storage Manager file space name
_serverdefObj.setTSMFileSpaceName("/dev/hda3");
// Set the identifier of Resource Manager objects within the TSM file space
_serverdefObj.setTSMFileSpaceInfo("RM1_Objects");
// Persist the information in the Resource Manager
_serverdefObj.update();
```

## DKRMVolumeDefSMS

The following new methods that are added to the DKRMVolumeDefSMS class provide a more flexible and granular way to control the migration of the objects:

Table 27. New methods in DKRMVolumeDefSMS class

| Methods          | Purpose                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| setHighThreshold | Sets a high threshold (threshold for disabling volume) where the volume is considered full. If this value is exceeded, the migrator might move objects until the low threshold setting (resume threshold for enabling volume) is reached to keep sufficient disk space available. The high threshold value (threshold for disabling volume) must be in the range of SMS_ALLOWED_MAX_HIGH_THRESHOLD and SMS_ALLOWED_MIN_HIGH_THRESHOLD . This value cannot be less than the low threshold value (resume threshold for enabling volume). |
| getHighThreshold | Returns the high threshold (threshold for disabling volume) where the volume is considered full                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| setLowThreshold  | Sets a low threshold (resume threshold for enabling volume) where migrator stops threshold migration and the volume becomes available again. This value must be greater than SMS_ALLOWED_MIN_LOW_THRESHOLD. This value cannot be greater than the high threshold value (threshold for disabling volume).                                                                                                                                                                                                                               |
| getLowThreshold  | Returns the low threshold                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

The following methods in the DKRMVolumeDefSMS class are deprecated:

#### **setThreshold**

Deprecated. Use setHighThreshold and setLowThreshold methods instead.

#### **getThreshold**

Deprecated. Use getHighThreshold and getLowThreshold methods instead.

See the Javadoc for more information.

### **Example**

The following example demonstrates the use of the setHighThreshold method in the DKRMVolumeDefSMS class:

```
DKRMConfigurationMgmtICM _rmCfgMgmt =
new DKRMConfigurationMgmtICM(dsICM);
DKResourceMgrDefICM _resMgrDef =
 (DKResourceMgrDefICM)_rmCfgMgmt.retrieveResourceMgr(rmName);
// Connect to resource manager
DKRMStoreSMS _rmStoreSMS = new DKRMStoreSMS(_resMgrDef);
_rmStoreSMS.connect(_resMgrDef, "");
DKRMVolumeDefSMS volume = DKRMVolumeDefSMS(_rmStoreSMS);
// Set high threshold value
volume.setHighThreshold(50);
// Set low threshold value
volume.setLowThreshold(10);
// Persist the information in the Resource Manager
volume.update();
```

### **DKRMControlServiceSMS**

There is only one significant method, DKRMConstantSMS.CONTROL\_SERVICE\_RESPONSE control in the DKRMControlServiceSMS class:

```
public DKRMConstantSMS.CONTROL_SERVICE_RESPONSE control
(DKRMConstantSMS.CONTROL_SERVICE_NAME service,
 DKRMConstantSMS.CONTROL_SERVICE_ACTION action)
```

The DKRMConstantSMS.CONTROL\_SERVICE\_RESPONSE control method takes a service name and action as input parameters and sends an order to the resource manager to perform the action specified on the service indicated. The resource manager sends a response back that indicates the result or the status of the action performed.

See the Javadoc for more information.

### **Example**

The following example demonstrates the use of the method in DKRMControlServiceSMS class:

```
\DKRMConfigurationMgmtICM _rmCfgMgmt =
new DKRMConfigurationMgmtICM(dsICM);
DKResourceMgrDefICM _resMgrDef =
 (DKResourceMgrDefICM)_rmCfgMgmt.retrieveResourceMgr(rmName);
// Connect to resource manager
DKRMStoreSMS _rmStoreSMS = new DKRMStoreSMS(_resMgrDef);
_rmStoreSMS.connect(_resMgrDef, "");
DKRMControlServiceSMS controlService = DKRMControlServiceSMS(_rmStoreSMS);
```

```

// Get the status of migrator
DKRMConstantSMS.CONTROL_SERVICE_RESPONSE response =
 controlService.control(DKRMConstantSMS.CONTROL_SERVICE_NAME.migrator,
 DKRMConstantSMS.CONTROL_SERVICE_ACTION.status);
System.out.println("Response = " + response);

```

---

## 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 include 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 contain 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 where 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. In the window are fields 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 part and enters some values for the attributes. The police report is a document part 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. You can 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

To create a document item type, follow the steps given in the examples below.

#### 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();
```

#### Creating a document item type example: 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.

### Creating a document example: 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.

**Important:** 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();
```

## Creating a document example: C++

1. Create a new DDO of type DocModelTest and semantic type

```
DK_CM_DOCUMENT
```

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
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");
```

2. 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;
```

3. 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);
}
```

4. Add the DDO to the datastore

```
ddoDocument->add()
```



For more information, see the SDocModelItemICM sample.

## Updating a document

The steps in this section 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.

### Updating a document example: 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. 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();
```

### Updating a document example: C++

1. Update the attribute value for the document item.

```
DKString attrValue = "New Value";
short dataId=ddoDocument->dataId
 (DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument->setData(dataId,attrValue);
```

2. Access the document's parts collection



```

// Access the DKParts Collection
DKParts* parts = null;
// Document's parts
dataId = ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
DK_CM_DKPARTS);
if(dataId == 0)
{ dataId = ddoDocument->addData(DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS);
parts = new DKParts();
ddoDocument->setData(dataId, parts);
}else{
parts = (DKParts*)(dkCollection*)ddoDocument->getData(dataId);
if(parts == null)
{parts = new DKParts();
ddoDocument->setData(dataId, parts);
}
}
}

```

### 3. 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);

```

### 4. Update the DDO information in the datastore.

```

ddoDocument->update();

```

For additional information, see the *SDocModelItemICM* sample.

## Retrieving and deleting a document

To retrieve a document with parts, use retrieve methods just like any other item.

For the document parts (DKParts collection), there is a difference between listing parts and retrieving data for the parts (including attributes and resource content). First, you must list the parts by populating the DKParts collection with blank DDOs and their completed PIDs that represent the parts data. However, you can also request additional parts data, including attributes and resource content within the same call to optimize retrieval. Furthermore, you can always make a subsequent call to retrieve attributes or other information for the parts themselves. You can pass the DKParts collection as a whole directly to multi-item retrieve (*DKDatastoreICM::retrieveObjects(dkCollection,DKNVPair[])*) because DKParts is a *dkCollection* which contains DKDDOs (as subclass *DKLobICM*). Alternatively, you can invoke single-item retrieve on any part individually.

If you want to list the parts, request *DKRetrieveOptionsICM::partsList(true)*. If you want the parts attributes (including system resource attributes), also include *DKRetrieveOptionsICM::partsAttributes(true)*. If you want the parts resource content, also include *DKRetrieveOptionsICM::resourceContent(true)* (single-item retrieve only). There are very important performance considerations and more details fully explained in the *Application Programming Reference* for the retrieve options. See the *DKRetrieveOptionsICM* documentation for each option for more information.

To delete a document with parts, call *delete (ddo.del())* just like any other item. 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.

### Example: 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;
DKItemRelationDefICM itemRelation =
(DKItemRelationDefICM)item.retrieveItemTypeRelation(partId);
versionControlPolicy = itemRelation.getVersionControl();
```

### Example: 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
DKItemRelationDefICM * itemTypeRelation =
(DKItemRelationDefICM*)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<sup>®</sup> 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

You must consider a few things when you are designing transactions in your applications.

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

DB2 Content Manager supports check-out and check-in operations on items.

The check-out operation is called to acquire a persistent write lock for items. When an item is checked out by a user, other users can not update it although they can still retrieve and view it. You need to call the check-out operation prior to updating or re-indexing an item, regardless of the transaction mode (implicit or explicit) that you use. When you are done with the item, call the check-in operation to release the persistent lock and make the item available for other users to update. After you create an item, you have the option to keep it in checked out state to prevent other users from changing it until you are completely done with the work. If you check-out (or check-in) an item using an explicit transaction, the checkout is undone if the transaction is rolled back. If you check-out an item using an implicit transaction, the checkout is committed. It is the application's responsibility to check the item back in, using checkin options or methods.

## Processing transactions

The transaction scope can be controlled by a client API call, but it must be designed carefully.

To group a set of API calls into a transaction, you must build it 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.

**Important:**

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.

## Explicit transactions behavior in Version 8.4

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)
throws DKNotExistException, DKException, Exception
```

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)
throws DKNotExistException, DKException, Exception
```

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 `DKUserMgmtAPI` in the Application Programming Reference.



---

## 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 begin writing queries, you must understand the query language concepts, syntax, and grammar.

To efficiently traverse DB2 Content Manager's hierarchical data model, you compose queries using the DB2 Content Manager 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.

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

You can build queries that perform three main types of searches: parametric, text, and combined parametric and text. To run a query, you can use one of the following methods: evaluate, execute, or executeWithCallback.

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.



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.

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. See the Application Programming Reference for description of the query options in the `execute` method.

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.

For more information on query methods, see the SSearchICM sample. See the Application Programming Reference for the evaluate, execute, or executeWithCallback methods in DKDatastoreICM class.

---

## Applying the query language to the DB2 Content Manager data model

Viewing the library server as an XML document can help you understand the query language.

However, 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.

### Folders

Folder functionality provided in Content Manager through the DKFolder is stored as a set of links of the DKFolder(DK\_ICM\_LINKTYPE\_NAME\_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 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. DB2 Content Manager 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.

For information about DB2 Net Search Extender used in DB2 Content Manager Version 8.2 or higher, see DB2 Information Center.

**Important:** DB2 Content Manager Version 8.4 still supports Version 8 functionality such as the definition class DKTextIndexDefICM and the methods using DKTextIndexDefICM of DKItemTypeDefICM and DKAttrDefICM.

## Searching for object contents

An end user performing a search does not notice any difference when searching for objects stored in a resource manager, but system administrators must perform an additional step when 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. 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

To locate documents search on their text contents.

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 interface. You can also modify the default properties of the created text index by using interface. For more information on the APIs, see the Application Programming Reference or the SItemCreationICM 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 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.

Basic search syntax also allows for use of + and -, as well as for use of quoted phrases. Basic search is supported on IBM DB2 UDB. .

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

**Not specified**

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 uses a different underlying text engine than does the basic text search, so the search syntax will be different. Advanced text search supports features such 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 ), 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.

For more information on advanced text search, see the SSearchICM sample.

## Text search using UDF ICMFetchFilter

The user exit, ICMFetchContentExit (ICMFetchContentParms1 \*IFP) is available to enhance text search.

For more information, see dcmcm118.dita#dcmcm118.dita

## ICMFetchFilter timeout duration

Enabling and setting the timeout duration can help you avoid crash/hang.

During text index update on items, the NSE engine obtains the text for indexing by calling the ICMFetchFilter UDF. The ICMFetchFilter UDF calls the INSO conversion filter to convert the binary data (PDF, DOC, and so on) to plain text. Because these invocations are in a sequence, the NSE index update process might be blocked when INSO stops.

DB2 Content Manager Version 8.4 provides an **Enable filter timeout** check box in **Features** tab in Library Server Configuration window to configure ICMFetchFilter timeout duration.

If you select the **Enable filter timeout** check box, you can specify the filter timeout value in **Filter timeout** list. If the **Enable filter timeout** check box is cleared, the timeout feature is not enabled.

When INSO conversion stops, DB2 Content Manager logs the information in the NSE log table about the document that caused the INSO to stop.

**Important:** The UDF timeout does not apply to the following platforms: iSeries™, and zSeries where NSE is not available.

The following two methods are added to the DKLSCfgDefICM class:

### getUDFTimeout

Returns the timeout duration that controls how long the ICMFetchFilter UDF waits for its text conversion child process.

### setUDFTimeout

Modifies the timeout duration that controls how long the ICMFetchFilter UDF waits for its text conversion child process. If the timeout value is positive, the ICMFetchFilter UDF waits for the text conversion child process for the duration specified before stopping the conversion. If the timeout value is zero, the method invokes the text conversion directly within the UDF. A negative timeout value results in an error.

For more information, see the DKLSCfgDefICM class in *Application Programming Reference*

## Indexing and searching in a Unicode DB2 UDB library server database

Users have experienced problems with the way DB2 Content Manager handled text indexing of non-English document content when using a Unicode DB2 UDB library server database.

The problems stem from the way text from documents was extracted by the ICMFetchFilter UDF. The ICMFetchFilter UDF is used to extract text from non-text documents, such as Word or PDF. That text was returned in the operating system code page, rather than the database code page. The text must be in the database code page in order for it to be interpreted properly by the DB2 Net Search



Extender during indexing. If the code page did not match, only words that exist in both code pages would work. This could give the appearance that only English words existed.

Many of the problems are fixed by adding the **ICMCCSID** library server environment variable.

By setting the **ICMCCSID** environment variable, the operating system code page is no longer required to be the same as the database code page. A possible value of **ICMCCSID** is 1208. Setting the environment variable allows text indexing of multi-language documents for a Unicode database. Therefore, you can now use a Unicode DB2 UDB library server database to index and search the following:

- Documents that contain content in multiple languages
- Documents that are in different languages, but belong to the same item type, using a single UTF-8 text index

### Setting up your environment:

If your environment is not set up correctly you will not be able to search for words in multiple languages, nor will text indexing of document content work properly.

Before you proceed, you should have DB2 Content Manager 8.3 Fix Pack 2 or later. You must also ensure that your library server database is Unicode (code page 1208). When you installed DB2 Content Manager you were given the option to select a Unicode database. You can also verify by connecting to the database and issuing the following command: `db2 get db config`. The command dumps the entire database configuration. Look for the line that contains Database code page and verify that the value is 1208.

The following information contains the main tasks:

- Setting the **CCSID** to 1208 in the library server
  - Recreating text indexes
  - Ensuring that text documents are UTF-8 encoded
1. Set the **CCSID** to 1208 in the library server. To text index document content in a Unicode database, you must use **ICMFetchFilter** to retrieve content from documents. You must also ensure that all text indexes are created with the **CCSID** set to 1208. The **ICMCCSID** environment variable must also be set to 1208 in the library server environment. To accomplish this, complete the following steps:

**UNIX** Export the **ICMCCSID** environment variable and add it to the **DB2ENVLIST** by completing the following steps:

- a. In the `/home/db2inst1/sqllib/userprofile` file, ensure that the following lines are included in the order listed:

```
ICMCCSID=1208
export ICMCCSID
```

- b. In the `/home/db2inst1/sqllib/profile.env` file, ensure that following lines are included and ensure that the **ICMCCSID** is added to the **DB2ENVLIST**:

```
DB2LIBPATH=/usr/lpp/icm/lib
DB2ENVLIST='LIBPATH ICMROOT ICDLL ICMCOMP EXTSHM
CMCOMMON DB2LIBPATH ICMCCSID'
```

- c. Recycle the DB2 UDB instance by entering

```
db2stop force
db2start
```



## Windows

On a Windows DB2 Content Manager system, set the **ICMCCSID** as a system environment variable then stop and restart DB2 UDB.

2. Recreate text indexes. If you have item types with text indexes that already exist in a Unicode library server database, you must drop the text indexes and recreate them with the new CCSID. To recreate the indexes see the `ManuallyDropDb2CMTextindexes.SQL` script at the bottom of this topic. You will need to read the comments in the script and customize it to your environment before it can be used to recreate text indexes. Important: This procedure directly modifies the library server database. Therefore, you should back up your database and text indexes before you attempt to manually drop and recreate the text indexes.
3. Ensure that text documents are UTF-8 encoded. Ensure that all plain text documents are UTF-8 encoded when storing them. You must manually indicate that these documents are UTF-8 encoded. If you do not, text search will work fine, but the end users might not be able to view plain text documents that do not have this field set. For details on how to set the encoding field for the document, see *Language and character encoding for imported text documents* in *eClient* and Specify language and character encoding information when you import plain text documents.

To ensure that plain text documents are indexed properly, you must also convert any plain text documents that are in the local code page to UTF-8.

When you have completed all of the tasks above, text indexing of document content will work properly. The text will be retrieved in Unicode and then converted to the correct format that is required by DB2 Net Search Extender. This enables text indexing of documents which contain text from multiple languages, since the text is all processed in Unicode.

For new text indexes, search can be performed on words of different languages, subject to the limitations listed below. For existing text indexes, the index must be dropped and recreated as described earlier. If plain text documents are added to the same node type, there will be no code page conversion. The encoding of the plain text documents must be UTF-8 (code page 1208) to be text indexed properly. Many editors allow text files to be saved with this encoding.

Support for searching documents in multiple languages is not available for library server databases that are not Unicode. For databases that are not Unicode, text search is limited to searching characters that are in the code page of the library server. Converting from a non-Unicode library server database to a Unicode library server database is not supported by DB2 Content Manager, although this may be possible through a service engagement.

Text indexing and search of multiple languages for a Unicode database is supported using `ICMFetchFilter`, it is also supported when a text attribute is made text searchable (`CHAR`, `VARCHAR`, `CLOB`). This is not supported for an attribute text index which uses `ICMFilter` (`BLOB` attributes). The DB2 Content Manager Java API fully supports searching for words in multiple languages. Since Java represents strings in Unicode, the Unicode characters taken in as part of a search string in Java will flow to the content manager server with no issues.

The C++ API does not support full Unicode, though it can be used to access a Unicode library server. The C++ API uses a single code page, which is not Unicode. Therefore, it can only search using words which can be represented in the code page, which it is using.

For a list of formats supported by Stellent see the following: [http://www.stellent.com/stellent3/groups/mkt/documents/nativepage/outside\\_in\\_supported\\_platforms.pdf](http://www.stellent.com/stellent3/groups/mkt/documents/nativepage/outside_in_supported_platforms.pdf).

The issue addressed applies only to DB2 Content Manager 8 systems which use DB2 UDB. Note that the MULTI-LEXER cannot be used with document item types and use of the BASIC-LEXER does not provide the same range of language support as is available on the library server on DB2 UDB with APAR IO02595.

### **ManuallyDropDb2CMTextindexes.SQL script:**

There are 6 steps that must be done successfully before text indexes can be recreated.

```
-- This procedure manually drops an item type text index from CM 8. However,
-- only document item types can recreate the text index, once it is dropped.
-- For resource item types, once the text index is dropped, it can not be
-- recreated using the admin client.
--
-- This procedure directly modifies the library server database. Therefore, you
-- should back up your database and text indexes before you attempt to manually
-- drop and recreate the text indexes.
--
-- In this script, all of the updates and deletes are commented out.
-- You must customize and run this script and look at the output of the queries
-- and then come back and uncomment and customize updates and deletes.
--
-- Connect to the library server database
-- Change this command to fill in your database name, userid, and password
CONNECT TO ICMNLSDB USER icmadmin using password;

-- Step 1:
-- This query selects the item type name and index names for the item type.
-- The example below finds the index names for the item type UniText.
-- To find all item type indexes in the system, remove the last AND condition from
-- the query below. Then query will show text indexes for all item
-- types. Note that the item type index names always end with 'TIE'.
-- If a schema other than 'icmadmin' is used, this must be customized below
select a.keywordname AS CompTypeName, d.indexname AS IndexName, b.itemtypeid
from icmadmin.icmstnlkeywords a,
 icmadmin.icmstitemtypedefs b,
 icmadmin.icmstcompdefs c,
 icmadmin.icmsttextindexconf d
where a.keywordclass = 2 and
 a.keywordcode = b.itemtypeid and
 b.itemtypeid >= 1000 and
 c.itemtypeid = b.itemtypeid and
 c.componenttypeid = d.componenttypeid AND
 d.indexName LIKE '%TIE'
-- Comment out this line to see all text indexes
AND a.keywordname = 'Forum'
;

-- Step 2:
-- Manually drop the text index. Use the index name from the query above.
-- Just as in the connect, you must customize database name, userid, and password
-- db2text drop index icmadmin.ICMUU01033001TIE for text connect to ICMNLSDB
-- user icmadmin using password
```

```

-- Step 3:
-- Now delete the row from the CM table that holds the configuration of the
-- the text index. This is needed to allow the index to be recreated with a
-- new configuration. Note that this deletes all configuration for this index.
-- You will need to re-enter the configuration information when you recreate
-- the index. You may want to look at the index in the System Admin GUI to
-- see the configuration, before doing this step.
-- Here you must set the index name to the name selected in the first query.
-- DELETE FROM icmadmin.icmsttextindexconf WHERE indexname='ICMUU01033001TIE';

-- Step 4:
-- Now we need to tell CM this item type is no longer searchable. To do this
-- we must set a flag in the component type definition of the attribute which
-- has the text index. For item type text indexes, the attribute
-- is called TIEREF. To see the current value of the flag, do the following
-- query. You must set the component type that contains the index and the name of
-- of the attribute that contains the text index in the sample shown below.
-- For resource item types this is simply the name of the item type. For document
-- item types this name must be derived from the ItemTypeId that was returned
-- from the first query above. The name of the comp type for a document item
-- type is ICMPart + the item type id. This is ICMParts1009 in the example
-- shown below.
SELECT a.keywordname AS CompTypeName, ca.ATTRFLAGS, kw.keywordname AS
 AttributeName, ca.ComponentTypeId, ca.AttributeId, ca.attributegroup
 FROM icmadmin.ICMSTCompAttrs ca,
 icmadmin.icmstnlskeywords a,
 icmadmin.icmstnlskeywords kw
 WHERE
 a.keywordclass = 5 AND
 a.keywordcode = ca.componenttypeid AND
 -- To list data for all indexes, comment on the following line
 a.keywordname = 'ICMParts1011' AND
 kw.keywordclass = 1 AND
 kw.keywordcode = ca.AttributeId AND
 kw.keywordname = 'TIEREF';

;

-- Step 5:
-- Tell CM this item type is no longer text searchable. To do this you must
-- set the attr flags for the TIEREF attribute to 3 in the database.
-- You must customize the query below to set the component type id from the
-- previous query.
-- UPDATE icmadmin.ICMSTCompAttrs ca
-- SET ca.ATTRFLAGS = 3
-- WHERE ca.componenttypeid = 1033 AND
-- ca.attributeid = 20 AND
-- ca.attributegroup = 7
-- ;

-- Step 6:
-- Now commit all the changes
COMMIT WORK;

-- Disconnect from the database
CONNECT RESET;

-- Now that all this is done you can go into the System Admin GUI and recreate
-- the text index for this item type.

```

---

## Creating combined parametric and text search

There are 3 types of searches that can be performed on virtually any piece of an item or component, text within an item or component, or text within resource content.

A 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.

## Creating combined parametric and text search: Java

Combining parametric and text searches in Java involves 5 steps.

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. Use the DKNVPair array to package the options. Always specify the retrieve options (among any other query options) to request the data that you plan to use and enable optimizations.

```
DKNVPair parms[] = new DKNVPair[3];
String strMax = "5";
parms[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, 5);
DKRetrieveOptionsICM dkRetrieveOptions =
 DKRetrieveOptionsICM::createInstance(dsICM);
dkRetrieveOptions.baseAttributes(true);
parms[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE, dkRetrieveOptions);
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 SSearchICMAPI education sample in *IBMCMROOT/samples/java/icm*.

## Creating combined parametric and text search: C++

Combining parametric and text searches in C++ involves 5 steps.

1. Specify any query options. Always specify retrieve options to request the data that you plan to use and enable optimizations. In C++, the DKNVPair[] options array always has to have a special END element for the query operation to detect the end. For example, if you want to specify two options, the options array must have three elements where the last element is the special END element.

```

| DKNVPair* parms = new DKNVPair[3];
| //Allow a maximum of 5 items to be returned from the search
| pparm[0].(DK_CM_PARM_MAX_RESULTS, DKString("5"));
| //Specify what content is to be retrieved
| DKRetrieveOptionsICM* dkRetrieveOptions =
| DKRetrieveOptionsICM::createInstance(dsICM);
| dkRetrieveOptions->baseAttributes(TRUE);
| pparm[1].set(DK_CM_PARM_RETRIEVE, dkRetrieveOptions);
| pparm[2].set(DK_CM_PARM_END,(long) 0);

```

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 required. Therefore `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())
 << " (" << ((DKPidICM*)ddo->getPidObject())
 << " " << ((DKPidICM*)ddo->getPidObject())
 << " " << endl;
}

```

4. Clean up.

```

delete(iter);
delete[] parms;
....

```

For a complete sample and additional documentation, see the `SSearchICMAPI` education sample in `IBMCMROOT/samples/cpp/icm`.

---

## Example searches using the query language

To begin writing queries, you must understand the query language concepts, syntax, and grammar.

The sample data model provided and an XML document representation of the data model, including sample queries, will help you better understand the query language and get you started with writing queries.

The sample queries in the following sections are based on the query examples data model outlined in Figure 11 on page 185. See the data model illustration when you

review the sample queries.

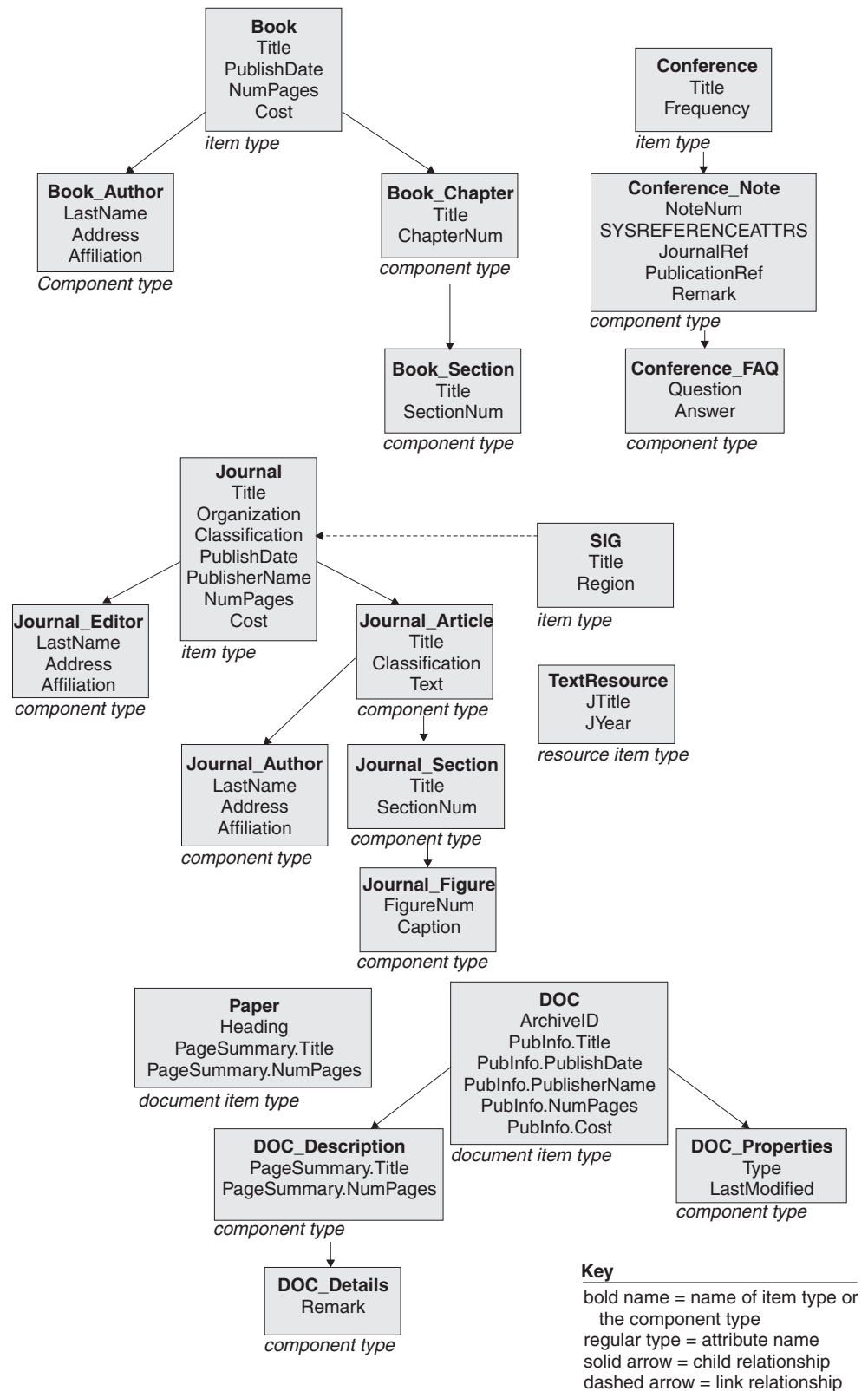


Figure 11. Query examples data model

For additional query syntax and examples based on an alternate data model, see the SSearchICM sample.

The XML document below is a representation of the data model in figure Figure 11 on page 185.

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>)
 <ICMCHECKEDOUT (STRING ICMCHKOUTUSER, TIMESTAMP ICMCHKOUTTS)
 </ICMCHECKEDOUT>
 ... (repeating <ICMCHECKEDOUT>)

</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>)

```

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

Examples of valid usage of UNION/INTERSECT/EXCEPT:

```
(//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.

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

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

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

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

## Examples of invalid usage

Examples of invalid usage of INTERSECT/EXCEPT:

```
/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 (=).

```
/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 (+).

## Query examples

To begin writing queries, you must understand the query language concepts, syntax, and grammar. These sample queries cover item types, text searches, arithmetic operations in conditions, traversal of links and references, checked out items, and querying by timestamp.

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<sup>®</sup> 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.

**Important:** When specifying attributes as query conditions, use the non-translated 15 character attribute name. Do not use the display name for your query conditions.

For more information on attribute names and display names, see the `SAttributeDefinitionCreationICM` and `SSearchICM` API education samples.

The sample queries provided in this section are based on the sample data model, Figure 11 on page 185, and the sample XML document.

### Example: Access to components

This query finds all journals.

```
/Journal
```

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: Access to attributes

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

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

The predicate `@NumPages = 50` evaluates to true for all journals that have the attribute `NumPages` set to 50.

### Example: 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%"])
```

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: 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]
```

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

## Example: 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
```

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: 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 => *
```

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.

## 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. This is an example of performing text search with the contains-text-db2 function. For the syntax supported by this function, see DB2 Information Center.

**Important:** 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 DB2 UDB Net Search Extender, which is used in this case to sort the resulting journal articles through SORTBY.

## Example: DB2

```
//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text-db2(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

## Text search (contains-text 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. 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: DB2

```
/Journal
[Journal_Article[contains-text-db2(@Title, " 'Design' |
 'Index' ")=1]]
SORTBY (@Title DESCENDING)
```

### Example: 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' "))
```

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.

### 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. 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. For the syntax supported by contains-text-db2 function, see the *IBM DB2 Universal Database: Net Search Extender Administration and User's Guide* in the DB2 UDB Information Center.

### Example: DB2

```
/TextResource[contains-text-db2(@TIEREF, " 'Java' & 'XML'
")=1]
```

### Example: 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 DB2 Information Integrator for Content..

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

### Example: 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: Traversal of references in the reverse direction

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

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

### Example: 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
```

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: 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: 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: 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: 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: System-defined attributes

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

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

### Text search on document model

This query finds all documents that contain the word XML in any one of its parts. 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: DB2

```
/Doc[contains-text-db2(./ICMPARTS/@TIEREF, " 'XML' ")=1]
```

### Example: 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: Document model (access to ICM Parts)

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

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

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: Existence of attributes

This query finds all root components that have a title.

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

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

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

### Example: 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: 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: 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: 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
```

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: Checked out items

This query finds all items of the Journal item type that are currently checked out.

```
/Journal [ICMCHECKEDOUT]
```

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: Checked out items by person

This query finds all items checked out by SMITH.

```
/Journal [ICMCHECKEDOUT/@ICMCHKOUTUSER = "SMITH"]
```

The value for ICMCHKOUTUSER must be entered in upper case in a query. Since the content servers store user IDs as uppercase, all queries must query for user IDs using upper case. All attribute data pertaining to user IDs must store them in upper case as well.

### Example: 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"]
```

---

## Using escape sequences in your queries

Proper handling of special characters ensures successful execution of queries and obtains the query results you want.

To support advanced features of the query language (like the wildcards % or \_ inside of text strings), escape sequences are used to differentiate between the cases when wildcards are treated as regular characters versus when they are given the special meaning of wildcard characters. For the user, it is important to know which characters are used as wildcards because wildcard characters, when intended to be treated as regular characters, must be preceded by an escape character. Escape sequences are also used to handle single and double quotes.

You need to add escape sequences when the strings used in queries contain either special characters (double-quote, apostrophe) or wildcard characters (percent sign, underscore, star, question mark) or a default escape character (a backlash). This handling is the simplest for strings used in comparison conditions and becomes a bit more involved for the LIKE operator and text search functions.

**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

Escape sequences do not always need to be used with comparison operators.

### Example: double quotation mark "

Precede your double quote with another double quote.

```
//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.

### Example: single quotation mark (apostrophe) '

You do not need to escape in this case.

```
/Book[@Title != "Uncle Tom's Cabin"]
```

## Using escape sequences with the LIKE operator

Escape sequences might not always be necessary with the LIKE operator, but usually they are.

### Double quotation mark "

Precede your double quote with another double quote.

```
//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. /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

Escape sequences are necessary when using quotation marks and wildcards in text searches.

### Double quotation mark "

Precede your double quote with another double quote.

#### DB2

```
//Journal_Article[contains-text-db2 (@Title,
" 'Analysis of '"The Time Machine'" %' ")=1]
```

Since the article's title contains the name of the book in double quotes, "The Time Machine", these internal double quotes need to be escaped.

### Single quotation mark(apostrophe) '

Precede the apostrophe with another apostrophe. A single apostrophe is not allowed in advanced text search because a set of apostrophes is used to enclose a term or a phrase. If an apostrophe appears inside a term, then the apostrophe needs to be escaped to differentiate it from the apostrophe that ends the term or the phrase.

#### DB2

```
/Book[contains-text-db2 (@Title,
" 'Uncle Tom''s Cabin' ")=1]SORTBY
(score-db2 (@Title, " 'Uncle Tom''s Cabin' "))
```

**Note:** Tom''s has two apostrophes.

#### DB2

```

/Book[contains-text-db2 (@Title,
" ('Greek' & 'Plato''s Symposium') & NOT ' Socrates' ")=1]
SORTBY (score-db2 (@Title, " ('Greek' & 'Plato''s Symposium')
& NOT ' Socrates' "))

```

**Note:** 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)

Escape sequences are necessary when using quotation marks and wildcards in text searches.

### Double quotation mark "

Precede your double quote with another double quote.

```

//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 enclosed 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"))
```

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 because 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.

Java and C++ escape sequences are handled as follows:

Example:

Query:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1]
```

### Example: Java

```
String query = "/Book[contains-text-basic (@Title, \"Why forgive\\?\")=1]";
```

### Example: 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

Row-based view filtering helps limit the amount of data retrieved for a given view. 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.

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.

**Important:** 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.

### Important considerations for using row-based filtering

If you use row-based filtering, there are some important usage guidelines that you should consider for better performance.

For operations that might return documents or items from multiple item types such as retrieving contents of a folder or worklist, depending on the ACL definition, the way in which DB2 Content Manager selects the subsets and follows the row-base filter conditions might not be clear and might affect the result sets.

If there are subsets (also known as views) with row-based filter conditions defined on the document item types, and if ACLs are defined to allow certain users to see the documents or items from more than one subsets, then DB2 Content Manager might choose a subset randomly and the retrieve action on the contents of a folder or worklist might not return all the documents that are expected to be in the result.

These problems can occur for the following operations:

- When retrieving documents using the Client for Windows or eClient, you open a folder or worklist.
- When you create a custom application (either Java or C++) using the DB2 Content Manager API that calls the `getWorkList` function or that retrieves a folder item by using the `DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND` or `DKConstant.DK_CM_CONTENT_ITEMTREE` methods (in DB2 Content Manager V8.2 and all fix packs, and DB2 Content Manager V8.3, fix pack levels earlier than Fix Pack 4) or the method `DKRetrieveOptionsICM::linksOutbound(true)` (in DB2 Content Manager V8.3 Fix Pack 4 and later).

In general, these problems can happen because of two main causes:

- One cause is that during the retrieve action, the user might have specified an active view where the filter condition filters out some documents to be retrieved, even though those documents are accessible according to the ACL definition. This problem happens because the manner in which the subset is chosen might not be what the user expects, for example:
  - For the Client for Windows or another application, users can explicitly choose the subset for the document item type.
  - For the eClient and other applications that do not allow the selection of the active view, a view with an ACL that allows the user to access the document is automatically chosen for the user by DB2 Content Manager. If there is more than one available view, the application chooses the view randomly. This view is usually, but not always, chosen in the order that the subset was created.

As a result, the active view that is chosen could be a wrong view that has a row-based filter condition, a condition that will filter out the expected documents.

- Another cause is that the retrieve operations are done in two steps. In the first step, the list of document IDs are returned based on the ACL setting for the subset. In the second step, the row-based filter is applied when each item ID is accessed. As a result, if the first step results in a large number of IDs, then the performance of the retrieve operation might be affected.

If you use row-based filters on subsets, your users should be aware of how these subsets can affect the result sets as described above. Carefully design your subsets, filter conditions, and ACLs accordingly.

**Tip:** Do not use a row-based filter condition if you already use autolinking to put documents in different folders. Instead, use the ACLs to restrict access to the folders if needed. If possible, carefully design your subsets and user groups so that your users can see all the documents that they are supposed to see with the selection of a single subset.

**Important:** If you rely on row-based filtering to filter out folder contents or worklists, this use of filtering might result in performance problems. This is because the retrieve first lists all the items regardless of filter condition, and then filters the items when the attributes are retrieved.

## Sample usage scenario

From a high-level perspective, row-based view filtering is 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.

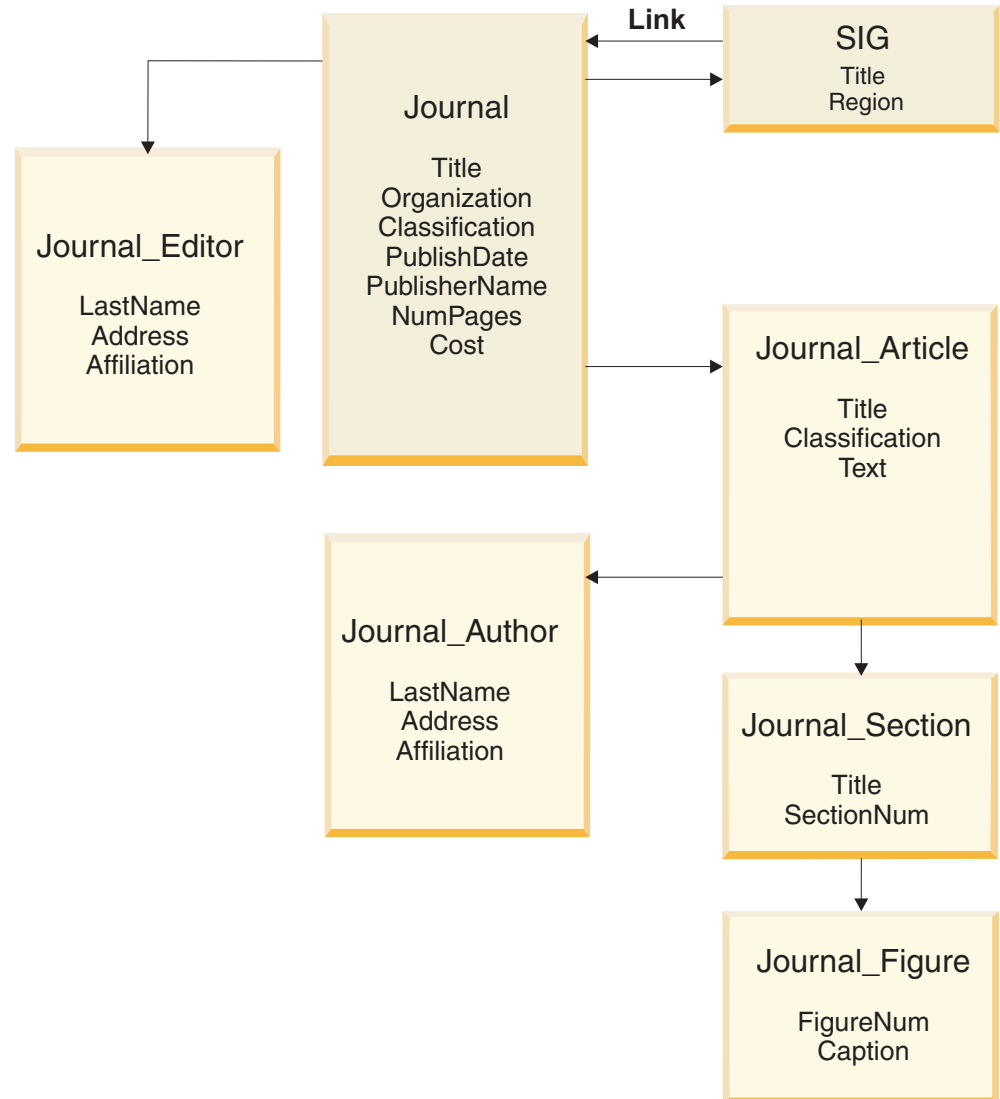


Figure 12. Sample data model for query row-based view filtering

Table 28. 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                       |

Table 28. Sample definition of views with filters (continued)

| User component type view | Base component type | Filter condition         |
|--------------------------|---------------------|--------------------------|
| MyJournal_Figure         | Journal_Figure      | @FigureNum = 5           |
| MyJournal_Author         | Journal_Author      | @Affiliation = "Almaden" |
| MyJournal_Editor         | Journal_Editor      | @Affiliation = "Almaden" |
| MySIG                    | SIG                 | @Region = "USA"          |

## Description of filtering behavior

How row-based view filtering behaves depends on the type of query that you want to perform. Row-based filtering in DB2 Content Manager query is applied on component type views that are 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.

### Root component 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 because the 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 because they are applied automatically based on which view you query.

**Restriction:** 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 nor MyJournal\_Figure filters are applied for the above `"/MyJournal"` query.

### Child component 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 white papers and other articles that might not be approved for public viewing are ignored.

**Restriction:** 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".

**Restriction:** 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 a result 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 wildcard characters

Because a wildcard character 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 character (\*) 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

Getting the best query performance depends on how you use filters and wildcard characters.

Follow these recommendations in your queries.



### **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 Organization attribute is a restrictive filter because only 1,000 components, out of a total of 1,000,000 components, have Organization = "IBM".

### **Minimize the use of wildcard characters in your queries**

Minimizing the use of wildcard characters in your queries is always a good idea. Specifically, because 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 wildcard characters. 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 character 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"]"`

Because 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 wildcard characters.

## **Database indexes for each filtered attribute**

Some database indexes are system defined, but some can be defined by a system administrator to improve performance.

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 create 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 was created on the Organization filter attribute for Journal, for example, a system administrator can open the 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, 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 queries and in other parts of the system.

For queries, there are cases for which the filters are not applied. Besides queries, 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**

Improve your query grammar with these keywords and literals in the Extended Backus-Naur Form (EBNF) notation.

#### **(\* 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, that is 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], 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 ;



---

## Routing a document through a process

The document routing APIs enable you to build new applications using document routing, or add document routing functionality into your existing applications.

DB2 Content Manager provides an integrated document routing service to help you route documents through a business process. Document routing provides 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, worklists, and work packages. The system administrator creates the work nodes, processes, and worklists 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. worklists define the work packages assigned to a user. You can have one or more worklists. Each worklist can include one or more work nodes. You can specify the order of the work packages in the worklist by priority, or date. You can also define the order of work nodes in the worklist.

When you retrieve worklists, 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 worklist 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 worklist
- Get the next item from a worklist
- Define, update, and delete a process
- Define, update, and delete work node
- Define, update, and delete a worklist

---

## Document routing features

To define document routing processes in DB2 Content Manager, you must use the provided graphical workflow builder.

### Graphical builder

A graphical and content-centric workflow builder is included in DB2 Content Manager . You can use it to model your document flows of business processes. 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.

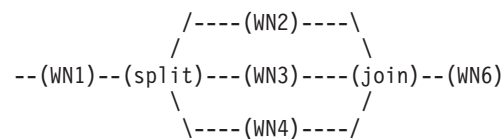
### XML of process (diagram) import and export

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.

### 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.

**Important:** 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.

### 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.

### Related information

-  Understanding compatibility with versions prior to Version 8.3
-  Understanding document routing classes
-  Document routing constants
-  Programming document routing user exits

## Understanding compatibility with versions prior to Version 8.3

If you make modifications to existing Version 8.2 processes or create new processes that take advantage of Version 8.3 functionality (that is, 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 the Version 8.3 library server that contains processes taking advantage of Version 8.3 document routing features.

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 rearranging the diagram layout, you need to reverify the diagram before you can save it.

## Changing the DOCROUTINGFREQ column in the ICMSTSYSCONTROL table for DB2

Using database commands, you must manually change the value for the frequency checking of updating the document routing work package attributes notify flag and suspend flag.

To define the document routing frequency checking value on DB2, complete the following steps:

1. Log on with a user ID that has at least db2admin authority. At a DB2 command prompt window, type the following: `db2 connect to dbname user userID using password` where *userID* is a valid DB2 user ID.
2. Determine the current DOCROUTINGFREQ setting. Enter the following: `db2 select docroutingfreq from icmstsyscontrol`.  
The default setting is 10, which means the minimum time the work package notify and suspend flags are updated is 10 minutes.
3. Specify the frequency. Enter the following: `db2 UPDATE icmadmin.ICMSTSysControl set DOCROUTINGFREQ = x where LIBRARYSERVERID =`

$y$  where  $x$  is the new document routing frequency checking value, and  $y$  is the library server ID value. The ID value is defined during installation and the typical value is 1.

4. Enter the following at the command prompt: `db2 connect reset`

---

## 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 *Application Programming 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 8.4 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 8.4 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 29 for all possible combinations.

*Table 29. 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          |

Table 29. Combinations that the add(DKProcessICM process) and update(DKProcessICM process) method can save (continued)

| Process state | Diagram definition | Route definition | Can be saved |
|---------------|--------------------|------------------|--------------|
| 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.

**Tip:** 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**

In Java and C++ programming, the DKDocRoutingServiceICM class provides the core routing services like starting, terminating, continuing, suspending, and resuming a process. The DKDocRoutingServiceMgmtICM object is a helper class that provides methods to manage DKProcessICM, DKWorkNodeICM, and DKWorkListICM and the meta-data required to define the objects.

The examples below demonstrate how to create a document routing object.

### **Example: Java**

```
// The DKDocRoutingServiceICM class:
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);

// Retrieving the DKDocRoutingServiceMgmtICM from DKDocRoutingServiceICM has
// the benefit of reducing cache footprint by sharing definition objects
// between these two classes.
DKDocRoutingServiceMgmtICM routingMgmt=
 routingService.getDocRoutingServiceMgmt();
```

### **Example: C++**

```
// The DKDocRoutingServiceICM class:
DKDocRoutingServiceICM* routingService = new
 DKDocRoutingServiceICM(dsICM);

// 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.

### Example: 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);
```

### Example: 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

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 listWorkNodeNames method lists work node names in the library server.

The listWorkNodes method returns a collection of DKWorkNodeICM objects representing work nodes in the library server.

### Example: 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());
}
```

### Example: 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 the requirements to meet before the process can resume or advance past a place in 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.

### Example: 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
// managment object
routingMgmt.add(collectionPoint);

```

### Example: 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 List of Requirements (Resume List).
resumeList->addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement->setFolderItemTypeName("book");
resumeRequirement->setRequiredItemTypeName("AnotherItemType");
resumeRequirement->setQuantityNeeded(1);
resumeList->addElement(resumeRequirement);

// 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
// managment 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 `S DocRoutingDefinitionCreationICM` sample.

## Defining a worklist

A worklist allows an administrator or a user application to dynamically change work assignments without contacting end users.

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.

Worklists are defined by the system administration client or the APIs. When using both the system administration client and the APIs, a worklist must include one or more work nodes. When defining a worklist by using the APIs, if there are zero work nodes included in the worklist, a `DK_ICM_MSG_EMPTY_WORK_LIST` exception is thrown. When defining a worklist by using the system administration client, the user cannot save the worklist dialog unless there are one or more work nodes included in the worklist.

However, the library server supports a worklist with zero work nodes. This can occur when a work node is deleted. When a work node is deleted, the library server removes the work node from any worklist that it is defined in. If a worklist has zero work nodes, the library server returns all of the workpackages in the system when the worklist is retrieved.

### Example: Java

```

// Create a new worklist.
DKWorkListICM workList = new DKWorkListICM();
// Choose a name of 15 characters or less.
workList.setName("S_fillClaimWL");

```

```

workList.setDescription("worklist 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 worklist
workList.setMaxResult(100);
String[] wnNames = {"S_fillClaim"};
workList.setWorkNodeNames(wnNames);

//Add the new worklist definition to the document
//routing management object
routingMgmt.add(workList);

```

### Example: 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 worklists, 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.

### Example: 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());
}
```

### Example: 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

Although you can create document routing process definitions with 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.

A document routing process is the defined routes that a work package being routed follows. Multiple routing processes can reuse the same nodes and you can also use multiple routes between nodes.

### Example: 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);

```

## Example: 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.

### Example: 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");
```

### Example: 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.

### Example: Java

```
routingService.terminateProcess(workPackagePidStr);
```

### Example: 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.

### Example: Java

```
workPackagePidStr = routingService.continueProcess
(workPackagePidStr, "Continue", "icmadmin");
```

### Example: 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.

### Example: Java

```
dkCollection requirements = new DKSequentialCollection();
//Process will be suspended for 2 minutes.
routingService.suspendProcess(workPackagePidStr, 2, requirements);
```



### Example: 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 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.

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.

### Example: Java

```
routingService.resumeProcess(workPackagePidStr);
```

### Example: 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

Based on the system administration setting for the worklist, the owner field in the listWorkPackagePidStrings method can return the work package PIDs of all work packages in a specified worklist.

Table 30. listWorkPackagePidStrings API results based on owner

| System<br>administration<br>setting for a<br>worklist | owner=empty                                                              | owner=logged on<br>user ID                                               | owner = a user ID other<br>than the logged on user<br>ID                                                                              |
|-------------------------------------------------------|--------------------------------------------------------------------------|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| Filter on owner is<br>not checked                     | Returns all work<br>packages in the<br>worklist.                         | Returns all work<br>packages in the<br>worklist.                         | Returns all work packages<br>in the work list.                                                                                        |
| Filter on owner is<br>checked                         | Returns work<br>packages of the<br>logged on user ID<br>in the worklist. | Returns work<br>packages of the<br>logged on user ID in<br>the worklist. | Returns work packages of<br>the specified user ID in<br>the worklist if the<br>ICM_PRIV_ITEM_GET<br>_ASSIGN_WORK<br>privilege is set. |

## Example: Java

The following code sample shows how to list the PID strings for all the work packages in a specified worklist.

```
String[] workPackagePIDs =
 routingService.listWorkPackagePidStrings(workListName,processOwner);

// Print Work Package PIDs
System.out.println("Work Packages in worklist: (+workPackagePIDs.length+));
for(int i=0; i< workPackagePIDs.length; i++)
 System.out.println(" - PID: +workPackagePIDs[i]);
```

## Example: C++

The following code sample shows how to list the PID strings for all the work packages in a specified worklist.

```
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 worklist: (" << 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

A work package contains all the necessary information about a process and about the item that it is transporting. 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.

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`).

## Example: 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());

```

### Example: 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 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.

The following example shows you how to list document routing processes.

### Example: 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());
}

```

### Example: 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.

### Related information



Understanding compatibility with versions prior to Version 8.3

## Implementing ad hoc routing

The system administration client can be used to set up the work nodes, processes, and worklists.

Below is an ad hoc routing example procedure.

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.
4. At runtime, 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.
5. 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

Refining your search on document queries can get you the specific information you need.

### Example 1

Returns car documents whose associated work packages are active (not suspended).

```
| /Car[@VERSIONID = latest-version(.)
| AND @SEMANTICTYPE = 1 AND
| 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 AND
| 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 AND 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
| AND 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
| AND REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
| /ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID
| AND ../@WORKNODENAME = "UnderReview" AND @SUSPENDFLAG = 1]]
```

## Granting privileges for document routing

To perform document routing operations, you must have the appropriate privileges. By default, an administrator user ID, with all privileges, is able to create, update, or delete a document routing process work nodes and work lists.

```
| In the DB2 Content Manager document routing processes, work lists and work
| nodes are managed like items. To create, delete, or update these document routing
| objects, you are required to have the following privileges: ItemAdd,
```

,ItemSetUserAttrs, or ItemDelete. By default, an administrator user ID, with all privileges, will be able to create, update, or delete a document routing processes work nodes and work lists.

Each document routing process, work node, and work list belongs to a system defined item type. These system-defined item types are bound to the system-defined ACL, DocRoutingACL. DocRoutingACL does not have any rules, but by default, every user that has the ItemSuperAccess privilege belongs to this ACL. Therefore, only users that have ItemSuperAccess privilege in addition to ItemAdd, which is required to in their privilege set, such as ICMADMIN (or any IBM DB2 Content Manager administrator) can create a new process, work node, or work list.

To allow a user that does not have ItemSuperAccess privilege to create a new document routing process, work node, or work list, the user must be added to the DocRoutingACL, with a privilege set that contains ItemAdd. Also, the user must have ItemAdd in their privilege set to create a new document routing object.

To allow a user that does not have ItemSuperAccess privilege to update or delete document routing administrative objects, the user must be in the ACL of that particular object with a privilege set that contains the correct privileges. To update an object, ItemSetSysAttrs and ItemSetUserAttrs must be in the associated privilege set. To delete an object, ItemDelete must be in the associated privilege set.

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 31. 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:<br>set the priority<br>set the owner<br>set the resume list<br>set the duration for suspension | suspendProcess resumeProcess<br>setWorkPackagePriority<br>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<br>getNextWorkPackage<br>checkOutItemInWorkPackage<br>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<br>getNextWorkPackage<br>getCount<br>listWorkPackagePidStrings            |

Table 31. Document routing privileges (continued)

| Privilege           | Description                                                    | Related API     |
|---------------------|----------------------------------------------------------------|-----------------|
| ICM_PRIV_ITEM_ROUTE | Used to see if the user is authorized to route a work package. | continueProcess |

## Working with access control lists for document routing

When an ACL is defined for a document routing entity such as a process, work node, and 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 32. Access control lists and document routing

| Objects   | Related methods                                                                                                      | Privileges                                                                          |
|-----------|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Process   | startProcess                                                                                                         | ICM_PRIV_ITEM_ROUTE_START                                                           |
| Work node | continueProcess suspendProcess<br>resumeProcess terminateProcess<br>setWorkPackagePriority<br>setWorkPackageOwner    | ICM_PRIV_ITEM_ROUTE<br>ICM_PRIV_ITEM_ROUTE_END<br>ICM_PRIV_ITEM_UPDATE_WORK         |
| Worklist  | getNextWorkPackagePidString<br>getNextWorkPackage getCount<br>listWorkPackagePidStrings<br>checkOutItemInWorkPackage | ICM_PRIV_ITEM_GET_WORK<br>ICM_PRIV_ITEM_GET_ASGN_WORK<br>ICM_PRIV_ITEM_GET_WORKLIST |

## Programming document routing user exits

A document routing user exit routine 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 in the following situations:

- When a work package is created and started on a process.
- When a work package moves to a work node.
- When a work package leaves the work node.
- When a collection point reaches a specific overload limit.

When DB2 Content Manager calls a user exit routine, 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 routine, use the ICMUSERSTRUCT, as defined in the sample header file *IBMCMROOT /samples/server/exit/wxv2tue.h*. You can include this header when compiling your own function into a DLL file, for example, *WXV2UserExitSample.dll*.

Because all work node user exit routines are only supported as synchronous processes (as shared libraries), you should minimize the application's run time. Otherwise, the library server times out if the exit routine's running transaction takes too long. Alternatively, to run the user exit routine as an asynchronous process, you can program the exit routine DLL to spawn a process that continues the business transaction separately from the library server.

When working with document routing user exit routines on UNIX operating systems, ensure that the DB2 UDB fence ID has execute permission on both the user exit routine and the directory where the user exit routine is located.

A C sample file, including guidelines, usage examples, and documentation for the user exit routine is in the *IBMCMROOT /samples/server/exit/icmdruext.c* directory. The name of your function must begin with the string *WXV2* to differentiate it from functions that you created prior to DB2 Content Manager 8.3.

## Document routing constants

Document routing constants are contained (defined) in *DKConstantICM*.

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





---

## 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.

Prefetching 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 prefetch the object from the collection it resides in and place it in the collection that you specified as the prefetch collection at define time. Figure 13 on page 238 depicts the flow of a prefetch transaction:

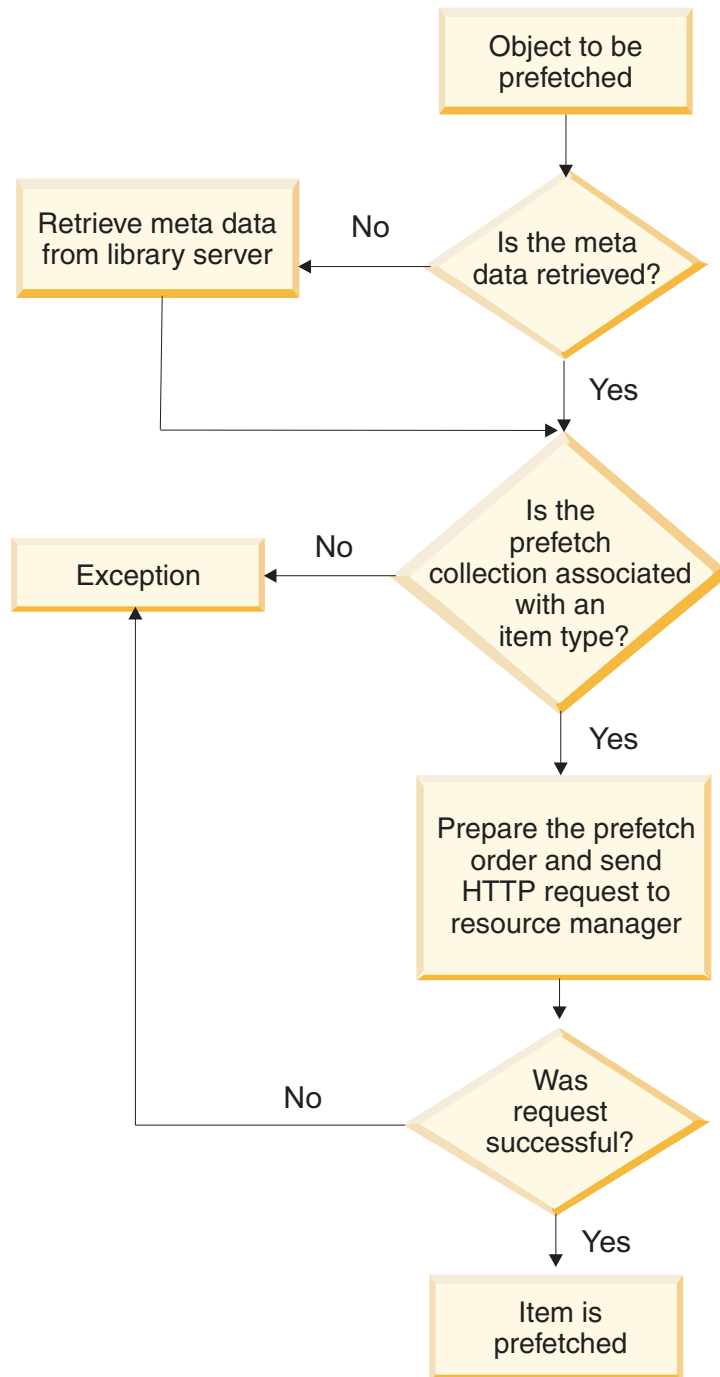


Figure 13. Prefetching objects

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.
2. The actual processing, which is copying an object from the source to the prefetch collection, is done by the ICM MOSAP asynchronous process. This process performs a select on the ICM RMCONTROL table for PREFETCHENABLED. If this value is set to 1, ICM MOSAP processes the entries in the ICM RMPREFETCH 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

This section discusses the table definitions related to prefetching.

*Table 33. ICM RMCONTROL*

|                 |          |           |
|-----------------|----------|-----------|
| PREFETCHENABLED | SMALLINT | NOT NULL, |
|-----------------|----------|-----------|

*Table 34. ICM RMPREFETCH*

|                  |               |          |
|------------------|---------------|----------|
| 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)       |          |

Table 34. ICMRMPREFETCH (continued)

|                          |              |          |
|--------------------------|--------------|----------|
| ITEMID                   | CHAR(26)     | NOT NULL |
| OPERATION                | VARCHAR(128) | NOT NULL |
| PREFETCHSTATE            | VARCHAR(128) | 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 extensibility 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.



---

## 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 DB2 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 4 on page 31 shows these classes.

*Table 35. Server type and class name terminology*

| Content server                                         | Class name     |
|--------------------------------------------------------|----------------|
| DB2 Content Manager Version 8.4                        | DKDatastoreICM |
| OnDemand                                               | DKDatastoreOD  |
| DB2 Content Manager for AS/400 (VisualInfo for AS/400) | DKDatastoreV4  |
| ImagePlus for OS/390                                   | DKDatastoreIP  |

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`. Example of concrete classes for this interface is `DKDatastoreDefICM`.

### **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. Example of concrete classes for the `dkEntityDef` interface is `DKItemTypeDefICM`.

### **dkAttrDef**

Defines methods to access attribute information and to create and delete attributes. Example of concrete classes for `dkAttrDef` is `DKAttrDefICM`.

### **dkServerDef**

Defines methods to access server information. Example of concrete classes for `dkServerDef` is `DKServerDefICM`.

### **dkResultSetCursor**

Creates a content server cursor that manages a collection of DDO objects.



### dkResource

Declares a common public interface for binary large object (BLOB) data.  
Example of concrete classes for dkResource is DKLobICM.

The data definition classes and their class hierarchy are represented in Figure 14:

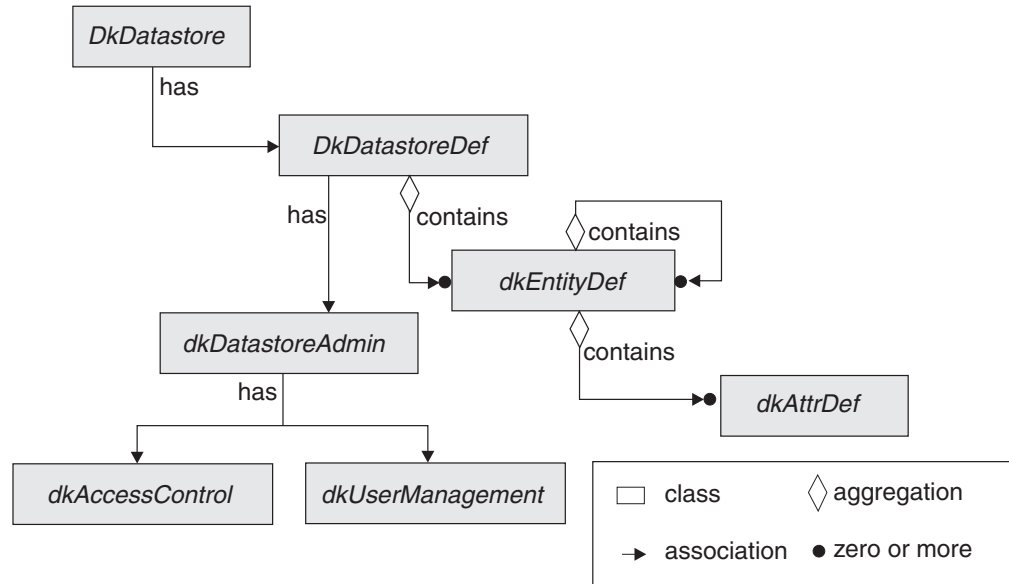


Figure 14. Data definition class hierarchy

## Working with OnDemand

DB2 Information Integrator for Content provides a connector and related classes for accessing content on OnDemand servers.

This is deprecated in DB2 Content Manager Version 8.4.

**Important:** OnDemand does not support Net Search Extender or Combined query.

### Representing OnDemand servers and documents

You represent a OnDemand content server using a DKDatastoreOD in an DB2 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

### **Important:**

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 Content Version 8.3, DKViews, DKFixedView, and DKResource replace DKViewDataOD, DKFixedViewDataOD, and DKResouceGrpOD respectively.

## **Connecting to and disconnecting from the OnDemand server**

This section discusses how to connect and disconnect 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.

### **Example: Java**

```
DKDatastoreOD dsOD = new DKDatastoreOD();
System.out.println("connecting to datastore ...");
dsOD.connect(ODServer, UserID, Password, "");
```

### **Example: 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.

### **Example: Java**

```
System.out.println("disconnecting from the datastore ...");
dsOD.disconnect();
dsOD.destroy(); // Finished with the datastore
```

### **Example: 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.

### Example: 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 - ");
}
...
```

### Example: 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:

### Example: 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");
...

```

### Example: 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.

### Example: 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();

```

### Example: 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.

### Example: 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");

```

### Example: 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.

### Example: 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

```

### Example: 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.

### Example: 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;

 if (pDO.protocol() == DK_XDO)
 {
 System.out.println(" dkXDO object " + i + " in collection");
 pXDO = (dkXDO)pDO;
 DKPidXDO pid2 = pXDO.getPidObject();
 System.out.println(" XDO pid string: " +
 pid2.pidString());
 //----- Retrieve and open instance handler for an XDO
 pXDO.retrieve();
 // pXDO.open();
 }
 }
 }
 else if (a != null)
 {
 System.out.println(" Attribute Value: " + a.toString());
 if (strDataName.equals("DKResource") ||
 strDataName.equals("DKFixedView") ||
 strDataName.equals("DKLargeObject"))
 {
 pDO = (dkDataObjectBase)a;

 if (pDO.protocol() == DK_XDO)
 {
 System.out.println(" dkXDO object ");
 pXDO = (dkXDO)pDO;
 DKPidXDO pid2 = pXDO.getPidObject();

```



```

 System.out.println(" XDO pid string: " +
 pid2.pidString());
 // Retrieve and open instance handler for an XDO
 pXDO.retrieve();
 // pXDO.open();

```

### Example: 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.

A sample configuration string would look like this:

### Example: Java

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

### Example: C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

A sample connect string would look like this:

### Example: Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
dsOD.connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

### Example: 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; click the **Stop Search** 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

Use visual beans for both direct and federated searches.

Three of the DB2 Information Integrator for Content visual JavaBeans, CMBSearchTemplateList, CMBSearchTemplateViewer, and CMBSearchResultsView, use DB2 Information Integrator for Content search templates. You can use these beans for federated searches by setting the CMBCConnection 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:

### Example: 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

```

## Example: 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 ImagePlus for OS/390

DB2 Information Integrator for Content APIs support a number of features when using ImagePlus for OS/390 content servers.

DB2 Information Integrator for Content APIs support the following features when working with ImagePlus for OS/390 content servers:

- Connecting and disconnecting from one or more 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:** ImagePlus for OS/390 does not support:

- Net Search Extender
- Combined query
- Workbasket and workflow

## Listing entities and attributes

After creating a content server for an ImagePlus for OS/390 content server as a DKDatastoreIP object and connecting, you can check the entities and attributes for the content server.

The following example lists all of the entities for an ImagePlus for OS/390 content server:

### Example: 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.

## Example: 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.

## Example: 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(" ");
 }
}

```

### Example: 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`.

### Example: 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(" ");
 }
 }
}
}

```



## Example: C++

```
// Method 2:
cout << "---List attributes using listEntityAttrs
function---" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
 docDef=(DKEntityDefIP*)(pIter->next()->value());
 //iterator returns DKAny*
 cout << " Document type : " << docDef->getType()
 << endl;
 cout << " Document type name: " << docDef->getName()
 << endl;
 DKSequentialCollection* pAttrCol =
 (DKSequentialCollection*)
 (dsIP.listEntityAttrs(docDef->getName()));
 if (pAttrCol == 0)
 {
 cout << "collection of entity attrs is null for entity "
 << docDef->getName()
 << endl;
 }
 else
 {
 int i=0;
 dkIterator* pAttrIter = pAttrCol->createIterator();
 while (pAttrIter->more())
 {
 i++;
 // ----- The iterator returns a pointer to DKAny
 attrDef = (DKAttrDefIP*)(pAttrIter->next()->value());
 cout << " Attr name [" << i << "] :
 " << attrDef->getName() << endl;
 cout << " Attr id :
 " << attrDef->getId() << endl;
 cout << " Entity name :
 " << attrDef->getEntityName() << endl;
 cout << " Datastore name:
 " << attrDef->datastoreName() << endl;
 cout << " Attr type :
 " << attrDef->getType() << endl;
 cout << " Attr restrict :
 " << attrDef->getStringType() << endl;
 cout << " Attr min val :
 " << attrDef->getMin() << endl;
 cout << " Attr max val :
 " << attrDef->getMax() << endl;
 cout << " Attr display :
 " << attrDef->getSize() << endl;
 cout << " Attr precision:
 " << attrDef->getPrecision() << endl;
 cout << " Attr scale :
 " << attrDef->getScale() << endl;
 cout << " Attr update ?
 " << attrDef->isUpdatable() << endl;
 cout << " Attr nullable ?
 " << attrDef->isNullable() << endl;
 cout << " Attr queryable?
 " << attrDef->isQueryable() << endl;
 cout << " " << endl;
 delete attrDef;
 } // end while
 delete pAttrIter;
 }
 delete pAttrCol;
 delete docDef;
```

```

 } // end while
delete pIter;
}
delete pCol;

```

## ImagePlus for OS/390 query syntax

The following example shows the query syntax for ImagePlus for OS/390.

### Example: Java

```

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

```

### Example: 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:

- Net Search Extender
- 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).

### 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.

### Example: C++

```
cout << "list index class(es)..." << endl;
pCol = (DKSequentialCollection*)
((dkCollection*)dsV4.listSchema());
pIter = pCol->createIterator();
i = 0;

while (pIter->more() == TRUE)
{
 i++;
 a = (*pIter->next());
 strIndexClass = a;
 cout << "index class name [" << i << "] - "
<< strIndexClass << endl;
 cout << " list attribute(s) for " << strIndexClass
<< " index class:" << endl;
 pCol2 =

(DKSequentialCollection*)((dkCollection*)
dsV4.listSchemaAttributes(strIndexClass));
 pIter2 = pCol2->createIterator();
 j = 0;

 while (pIter2->more() == TRUE)
 {
 j++;
 pA = pIter2->next();
 pDef = (DKAttributeDef*) pA->value();
 cout << " Attribute name [" << j << "] - "
<< pDef->name << endl;
 cout << " datastoreType - " << pDef->datastoreType
<< endl;
 cout << " attributeOf - " << pDef->attributeOf
```

```

<< endl;
cout << " type - " << pDef->type << endl;
cout << " size - " << pDef->size << endl;
cout << " id - " << pDef->id << endl;
cout << " nullable - " << pDef->nullable
<< endl;
cout << " precision - " << pDef->precision
<< endl;
cout << " scale - " << pDef->scale << endl;
cout << " string type - " << pDef->stringType
<< endl;
}

cout << " " << j << " attribute(s) listed for "
 << strIndexClass << " index class" << endl;
pCol2->apply(deleteDKAttributeDef);
delete pIter2;
delete pCol2;
}

delete pIter;
delete pCol;
cout << i << " index class(es) listed" << endl;
dsV4.disconnect();
cout << "datastore disconnected" << endl;

```

The complete sample application from which this application was taken (TListCatalogV4.cpp) is available in the samples directory.

## Running a query

The following example runs a query in DB2 Content Manager for AS/400, and processes the results.

### Example: 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...
 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.

### Example: 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...
 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());
 pDDBase = a;

 if (pDDBase->protocol() == DK_PDDO)
 {
 pDDO = (DKDDO*)pDDBase;
 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...
 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.

### Example: 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.

### Example: 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.



---

## Building DB2 Information Integrator for Content workflow applications (deprecated)

Using the DB2 Information Integrator for Content classes and APIs, you can create or extend your own applications to use the DB2 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.

This is deprecated in DB2 Content Manager Version 8.4.

---

### 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. When you are finished using the workflow service, you must disconnect by calling the `disconnect()` and the `delete()` functions.

The following example starts workflow services:

#### Example: Java

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

#### Example: C++

```
// ----- Create the strings for the name of the service, user ID
// ----- and Password
DKString wfsrv = "icmnlbdb";
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,"");
```

The following example disconnects the workflow services:

### Example: Java

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

### Example: 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:

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.

### Example: Starting a 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("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 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.

### 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();
}

```

### Example: 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`.

### Example: 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;
}
```

### Example: 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 either indefinitely or with a specific time. If you provide a null `DKTimestamp`, then DB2 Information Integrator for Content suspends the workflow indefinitely.

The following example shows how to suspend a workflow until a certain time.

### Example: 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);
}
```

### Example: 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.

### Example: 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();
}
```

### Example: 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`.

### Example: 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;
}

```

### Example: 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.

### Example: 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());

```

### Example: 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.

### Example: 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;

```

### Example: 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.

### Example: 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);

```

### Example: 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`.

### Example: 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;
}

```

### Example: 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 create actions that you can use in a workflow by using action objects (DKWorkflowActionFed objects). You define the actions and add them to action lists in DB2 Information Integrator for Content Administration.

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.

### Example

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 DB2 Information Integrator for Content workflow JavaBeans

Beans provide functionality for building workflow applications. For example, you can use 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.

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. You can use the JavaBeans in builders that support them (see the product documentation for your builder for additional information).

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 these components installed and configured before you can work with DB2 Information Integrator for Content Workflow JavaBeans.

- IBM WebSphere Application Server 4.0.4
- IBM DB2 Information Integrator for Content V 8.1, Fix Pack 1 and prerequisites
 

You must have installed DB2 Information Integrator for Content with the workflow option and an LDAP server to enable user import and user authentication.
- IBM Content Manager 8.1, Fix Pack 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

There are 9 steps you need to do to set up your data model.

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
- SystemDefineNLSTLang
- SystemBatchCompileACL
- SystemDefineMimeType
- SystemManageKey
- SystemDefineAttr
- 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
2. Create the user groups in the list below.
    - Workflow Participants
    - Content Publisher
    - Project Lead
    - Content Contributor
    - Domain Expert

**Note:** 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.

*Table 36. 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.

*Table 37. 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.

*Table 38. Item types*

| Name (case sensitive) | Item type classification | Access Control | Attributes     |
|-----------------------|--------------------------|----------------|----------------|
| WCM_Document          | Item                     | WCMACL         | WCM_Fields     |
| WCM_Folder            | Item                     | WCMACL         | WorkPackageACL |

**Notes:** 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.

*Table 39. 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.

Name: Simple Change Process Description: Process model for simple change process Access control list: WCMACL

*Table 40. 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.

Create the following users on your LDAP server and then import them into Content Manager:

*Table 41. 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" |

See the Creating users section 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.

7. (Optional) Add 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 CustomAttributeTool -h for more options.

You can download this tool from the IBM support Web site.

#### 8. (Optional) Add 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. Create a worklist. Create the worklist below with the characteristics shown.

Leave the default values if they are not specified.

Name: WCM\_WL (case sensitive)

Access control list: WCMACL

Nodes: Add all the work nodes that are listed in the process managers' routes (Table 40 on page 286).

Optional sample setup: Add the following work nodes from the **Node** tab:  
**Request Change, Review Request, Make Change, Review Change**

Document routing

Worklist

Attributes

Creating users

Creating users groups

Creating privilege sets

Creating access control lists

Defining collection point

Item types

## Using 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.

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

```
HttpSession session = request.getSession();
session.setAttribute("com.ibm.mm.beans.wcm.ICMServerName", "ICMNLSDDB");
```

In the example above, ICMNLSDDB is the datastore name.

### Example code snippets

There are common tasks that you can complete using the DB2 Information Integrator for Content workflow JavaBeans.

These 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);

```



---

## Building applications with non-visual and visual JavaBeans

You can use the visual and non-visual JavaBeans provided in DB2 Information Integrator for Content to build Java-based or Web client applications.

The DB2 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

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.

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

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 the JDK level required by the beans.

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 DB2 Information Integrator for Content beans, which are in `cmb81.jar`.

**Note:** To use the beans, you must have Java JDK 1.5.

The `%IBMCMROOT%\samples\java\beans` directory contains code samples of the non-visual beans, and the `%IBMCMROOT%\samples\java\beans\gui` directory contains a sample application written using the visual and non-visual beans.

## Using IBM Websphere Studio Application Developer

Use non-visual beans to build applications in WebSphere Studio Application Developer.

To build servlets and JSP pages by using non-visual beans in Websphere Studio Application Developer, complete 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\db2jcc.jar`  
`\SQLLIB\java\db2jcc_license_cu.jar`
3. If you plan on using the DB2 Information Integrator for Content servlets and JSP taglib, you must also specify the following files:  
`%IBMCMROOT%\lib\cmb servlet81.jar`  
`%IBMCMROOT%\lib\cmbtag81.jar`

For the tag library, you also need to import the taglib.tld file for DB2 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 your 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 DB2 Information Integrator for Content JavaBeans

You can invoke DB2 Information Integrator for Content JavaBeans either by calling them directly or by wiring instances of beans to other beans.

The beans in the DB2 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 DB2 Information Integrator for Content beans by using request and reply events. When using this method, remember the following:

- The `CMBCConnection` bean listens to connection request events and replies by firing connection reply events.
- The `CMBDataManagement` bean listens to data request events and fires data reply events in return.
- The `CMBSchemaManagement` bean 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 bean listens to workflow data request events and fires workflow data reply events in return.
- The CMBWorkflowQueryService bean listens to worklist request events and fires worklist reply events in return.

---

## Working with the non-visual beans

DB2 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 DB2 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 DB2 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 import capabilities. If you need batch processes for importing and exporting large amounts of data, you should use the connector interfaces.
- Complete server-specific functionality for all connectors. Most beans functions must work across all server types, and though the beans include some server-specific functionality for certain servers, the beans do not surface the full functionality of each server that is available from the Java APIs.

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 DB2 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 non-visual bean 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

Non-visual beans provide a number of features that enables the use of beans in JSP.

You can use the DB2 Information Integrator for Content beans in JSP because 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:** 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 DB2 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 DB2 Information Integrator for Content.
- Provide functionality to search and retrieve documents, simple items, and resource items in all of the repositories supported by DB2 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 DB2 Information Integrator for Content.
- Integrate and expose the functionality provided in the DB2 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

Non-visual beans categories include: datastore, helper, ancillary, XML services bean and helper classes, workflow, document services, and others.

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 back-end 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 DB2 Information Integrator for Content advanced workflow processes.

#### **CMBDocRoutingDataManagement and CMBDocRoutingQueryService**

Used to work with DB2 Content Manager Version 8 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 DB2 Content Manager Version 8 repository, a CMBEntity represents both item types and child component definitions, while for an OnDemand repository, a CMBEntity represents an application group. CMBEntity is a helper class for CMBSchemaManagement.

#### **CMBAttribute**

Represents attribute definitions in the repositories. CMBAttribute is a helper class for CMBSchemaManagement.

#### **CMBSearchTemplate**

Represents a federated search template. CMBSearchTemplate is a helper class for CMBSchemaManagement.

#### **CMBSTCriterion**

Represents a search criterion that is a part of a federated search template. CMBSTCriterion is a helper class for CMBSchemaManagement.

#### **CMBItem**

Represents instances of documents, resource items and non-resource items. CMBItem is a helper class for CMBDataManagement.

**CMBOject**

Represents instances of resource items, base parts and notelog parts. CMBOject is also used to represent BLOB attributes. CMBOject is a helper class for CMBDataManagement.

**CMBAnnotation**

Represents instances of annotation parts for DB2 Content Manager Version 8 repositories and notes for OnDemand repositories.

**CMBPrivilege**

Provides the functionality required to retrieve privilege related information from a CM or DB2 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 pool MMBConnection beans. MMBConnection beans wrap DKDatastore instances, and these DKDatastore instances can be expensive to create. Using the MMBConnectionPool allows re-use of the MMBConnection bean instances, and the DKDatastore instances that they wrap.

**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, exporting item type definitions as XML schemas, and performing document routing operations. The XML bean leverages the XML support in the API for item import, item export, and schema export operations.

**CMBXMLServices**

This bean is the underlying component of Content Manager Web services and shares its messaging format with the Web services. The CMBXMLServices bean reads configuration from the cmbxmlservices.properties file in %IBMCMROOT%/cmgmt. The XML requests sent to this bean must conform to the schema defined in the cmbmessages.xsd file in %IBMCMROOT%/config.

**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 contains the resource content for a resource object or document part that is needed to fulfill the request operation.

### **Workflow beans**

The workflow beans provide workflow services. The workflow services provided by the DB2 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 DB2 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 `CMBCConnection` 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 DB2 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 sent 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 DB2 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 DB2 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 DB2 Information Integrator for Content.

Consider the following 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 DB2 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 been connected or disconnected, so that the beans can take appropriate action.

**Important:** The connection reply events are only sent when the datastore is connected or disconnected as a result of a method called on the CMBConnection bean instance.

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.

## The CMBConnectionPool bean

The CMBConnectionPool bean maintains a pool of live and connected CMBConnection instances.

CMBConnection beans wrap DKDatastore instances. Therefore, the pool is in effect a pool of DKDatastore objects. Because the datastore instances held by the pool are connected and remain connected for as long as they are managed by the pool, each instance is tied to the specific user ID and password that were used to create the datastore instance. The connection pool will return only a pre-existing connection object from the pool if the user ID and password in the new connection request exactly matches the user ID and password data. If no existing connection is available, then the pool creates a new connection for that user ID. It does not attempt to disconnect or reuse one of the idle connections in the pool that are connected with another user's credentials.

### Connection pool use

You can use the connection pool when your application is designed such that in the course of using the application, the same user ID makes multiple requests for a connection object, and returns the connection to the pool after performing a few tasks with the connection.

If your application is designed such that a single user ID never requests a connection more than once per session, there is no benefit to using a connection pool.

### Thread queuing

The CMBConnectionPool bean has two options when no matching connection is found in the pool for a connection request and there is no more room in the pool

to create a new connection object. This behavior is controlled by the `maxConnectionsBehavior` property. The default value for this property is `CMBConnectionPool.CMB_MAX_CONNECTIONS_ERROR`. When no connection is found in the pool that matches the credential information supplied in a `getConnection` request, and if there is not enough room in the pool to create a new connection for the user, the `CMBConnectionPool` returns an error. The other possible setting for this property is `CMBConnectionPool.CMB_MAX_CONNECTIONS_QUEUE`, which enables thread queuing. When no matching connection is found and there is not enough room in the pool to create a connection, the thread making the connection request is put into a wait state. The thread waits until one of following situations occur:

- A connection with matching credentials is freed back to the pool by another thread. In this case, the first waiting thread with credentials that match those of the connection being freed is activated, and the connection that was just freed will be returned to that thread.
- The thread wait time is exceeded. You can specify the maximum amount of time you want a thread to wait for an available connection by setting the `maxThreadWaitTime` property on the `CMBConnectionPool`. If your thread waits for longer than the wait time and there is still no connection available to fulfill the `getConnection` request, the request will return with an exception.
- A connection that is being freed to the pool must be destroyed. When putting a connection back into the free pool violates one of the free connection tuning parameters, that connection is destroyed instead of being returned to the pool. If any threads are waiting, the first waiting thread is activated and a new connection is created for that thread.
- One of the connections in the free pool times out and must be destroyed. You can specify the maximum amount of time that a connection can remain idle in the free pool by setting the `maxWaitTime` property on the `CMBConnectionPool`. Any connection that remains idle in the free pool for longer than the `maxWaitTime` is destroyed. At the time the expired connection is destroyed, if there are any threads waiting, a new connection is created for the first waiting thread.

Depending on your application's use and the tuning settings on the pool, it is possible for a thread to wait indefinitely for a connection, but never receive a new connection. This might not necessarily represent an error in the pool but it might indicate a situation where the tuning parameters set on the pool are not adequate to ensure that all users can get a connection in a reasonable amount of time. For example, if you set the `maxConnections` property to 10 and 10 users get connections from the pool but never return them, any further connection requests from the pool wait until they time out, because no connections are being freed, and there is no room in the pool to create more connections.

## **CMBConnectionPool tuning options**

Maintaining a pool of connected `DKDatastores` can use up a great deal of memory. The `CMBConnectionPool` bean provides many properties to help users tune the memory usage of the pool to best meet their application's specific needs. The optimal settings for these parameters will vary according to the specific environment and use of the application. The fastest way to get a connection from the pool is to find an existing connection in the free pool that can be re-used for the `getConnection` request. Users must strive to set the tuning parameters to encourage the maximum amount of connection re-use and to minimize the need to destroy connections and create new ones to fulfill a `getConnection` request.

The following properties of the connection pool are used to control the pool's overall memory footprint and to manage the amount of time each thread must wait for a connection.

#### **maxConnections**

The `maxConnections` property controls the total number of connections managed by the pool. This is the total of all connections in use and all the available connections being held in the free pool. This setting can be used to prevent the pool from growing too large for the memory limits of your system. If there is no maximum value of the total number of connections managed by the pool, you can set the value to `CMBCConnectionPool.CMB_NO_MAX`. The default value is 100. Set the `maxConnections` value to at least 2 to 3 times the number of different user IDs that will be accessing the pool. If you have a single user ID that is shared by many users, then set the `maxConnections` value to 2 to 3 times the number of different users sharing that user ID. Setting the `maxConnections` property value too low will result in poor performance, increased failures in `getConnection` requests, and possibly long waits for users when thread queuing is enabled.

#### **maxConnectionsPerUserid**

The `maxConnectionsPerUserid` property controls the total number of connections per user ID managed by the pool. The default value is `CMBCConnectionPool.CMB_NO_MAX`. This is the total of connections connected with that user ID that are currently being used plus the available connections in the free pool that are connected with that user ID. This setting is used to prevent one user ID from filling the pool with connections for that user ID, leaving little room for requests from other user IDs. When pool usage is extremely unbalanced, such as 80 percent of requests coming from one user ID, the other 20 percent come from a variety of user IDs, the other user IDs might experience more failures to get connections, or long waits for connections if thread queuing is enabled. Limiting the number of connections that any single user ID can create is one way to ensure better response times for the other user IDs. However, such extremely unbalanced usage of the pool is not typical, and therefore this parameter is not typically helpful.

**Important:** Setting the `maxConnectionsPerUserid` value to anything other than `CMB_NO_MAX` has a detrimental impact on performance because the overhead of scanning the pool to enforce this maximum is significant. You should use this setting only if testing on your system has proven that the benefits of limiting one user ID from filling the pool outweigh the performance hit of enforcing this setting.

#### **maxFreeConnections**

The `maxFreeConnections` property controls the total number of connections allowed in the free pool, which is the pool of connections that returned to the `CMBCConnectionPool` by calling the `freeConnection` method. This setting is used to control the amount of memory that is taken up by the pool when the pool is idle. After the limit in the free pool reached, freeing additional connections results in connections in the free pool being destroyed to avoid going over the limit. Setting this value too low can negate the usefulness of the pool by causing too much destroying and re-creating of connections and preventing reuse of connections. This parameter can be useful to limit the amount of memory that is used by the pool when the pool is idle. You might choose to set a very high limit or no

limit at all on the maximum number of connections so that all users can be served during peak usage. However, after the peak usage has passed, the pool holds a large number of connections in memory. This parameter can help you control how many unused connections the pool holds onto, allowing you some means of reducing the memory footprint of the pool. Do not set the limit too low or you might negatively impact the performance of the pool.

#### **maxThreadWaitTime**

The `maxThreadWaitTime` property applies only when thread queuing is enabled. This property controls the maximum amount of time any thread will wait for a connection to be made available in the queue. After the wait time has elapsed, if there are still no connections available, the `getConnection` request for that thread will return with a `CMBGetConnectionFailedException`, with an error ID indicating timeout. The default value is `CMBConnectionPool.CMB_NO_TIMEOUT`, which means that by default, threads wait indefinitely for a connection to become available.

#### **timeOut**

The `timeOut` property controls how long a connection is allowed to sit idle in the free pool before it is disconnected and destroyed. By default, the value is `CMBConnectionPool.CMB_NO_TIMEOUT`, which means that connections can remain in the free pool indefinitely. The setting is enforced by a timeout thread that polls at regular intervals to see how long each free connection has been idle. The `timeOut` property is most useful when it is set to coincide with the amount of time the underlying database connection can remain idle before it becomes invalid. If you are connecting to a datastore that is implemented such that you do not have any concerns about the underlying database connection ever becoming invalid, then you do not need to use this setting. If you are pooling Content Manager V8 connections and you have Websphere Application Server connection pooling enabled for the datastore, you should not set this property to anything other than `CMBConnectionPool.CMB_NO_TIMEOUT`. Setting the property to any other value causes performance problems and has no benefit because enabling Websphere Application Server connection pooling in the datastore eliminates the underlying database connection timing out.

## **Display names**

In DB2 Content Manager Version 8.4 connector, you can translate display names for an entire set of languages.

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).

To specify which language's display names that you want to appear in your application, call `CMBConnection.setLanguageCode(String languageCode)`. The display names that are returned by the beans will be translated into the language you specify in this `languageCode` property.

Both `CMBEntity` and `CMBAttribute` contain three methods to retrieve the name and display name:

#### **getNonDisplayName()**

Always returns the real, non-display name.

### **getDisplayName()**

Always returns the display name translated into the language specified by the CMBConnection bean's languageCode property.

### **getName()**

If displayNameEnabled equals false (this is the default), then getName() returns the non-display name. If displayNameEnabled is set to true, then getName() returns the display name (translated into the language specified by the CMBConnection bean's languageCode property).

## **Tracing and logging in the beans**

You can enable tracing on all session-wide beans in the DB2 Information Integrator for Content beans layer.

Enabling tracing on an instance of the CMBConnection bean also enables tracing on any DB2 Information Integrator for Content bean obtained from this connection bean, including schema management, data management, query service, and workflow beans. For the tracing in beans to take effect, you must enable beans tracing in the cmblogconfig.properties in %IBMCMROOT%\cmgmt\connectors. Beans trace settings are located in section 6 of this file.

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 a beans trace log, located by default in %IBMCMROOT%\log\beans. The location and name of this log can be modified by changing settings in the cmblogconfig.properties file.

## **Understanding properties and events for non-visual beans**

Each non-visual bean provides a number of properties and events.

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**

These properties are set and read only through the setter and getter methods on the beans. No events are fired when setting these properties.

Each non-visual bean provides the following:

- Notifies listeners of events generated by this bean. For example, the CMBConnectionBean will automatically fire CMBConnectionReplyEvents to all registered listeners when the connect() or disconnect() methods are called on the bean.



- The ability to listen to events in which this bean is interested. For example, the CMBSearchResults bean has a built-in listener for CMBSearchReplyEvents, so once it is added as a listener to an instance of the CMBQueryServiceBean, which is the source of the CMBSearchReplyEvents, it will be able to automatically process the searchReplyEvents fired by that bean.

## Building an application using non-visual beans

You can use non-visual beans to build an application.

### 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 and CMBXMLServices beans. The complete sample application from which this example was taken (DemoSimpleAppl.java) is available in the Samples directory. The sample application shows how to:

1. Connect to the DB2 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.

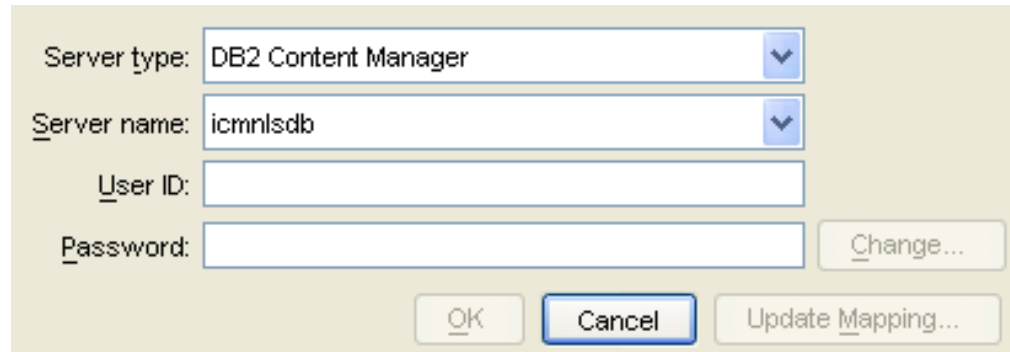
Visual beans perform basic tasks that are common to many applications, such as logging on, searching, displaying and viewing results, updating documents, and viewing version information.

Each visual bean has a Connection property. This property must reference an instance of CMBCConnection, the nonvisual bean that maintains the connection to the content servers. Any application built with the DB2 Information Integrator for Content visual beans must also contain an instance of the CMBCConnection nonvisual bean.

### CMBLogonPanel bean

This bean displays a panel to log in to DB2 Information Integrator for Content or to content servers such as DB2 Content Manager Version 8.4. It also provides the window where federated users can modify the user IDs and passwords on the content servers.

The CMBLogonPanel bean (see Figure 15 on page 307) displays a window that lets users login to a content server, update user mappings, and change a password.



Server type: DB2 Content Manager

Server name: icmnlbdb

User ID:

Password: Change...

OK Cancel Update Mapping...

Figure 15. CMBLogonPanel bean window

In the CMBLogonPanel bean, when a user clicks **Change**, the Change Password window appears (see Figure 16.) The user enters the old password, and enters the new password twice.



**Change Password**

Please enter a new password.

New password:

Confirm password:

OK Cancel Apply

Figure 16. 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 17 on page 308). When you click **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.



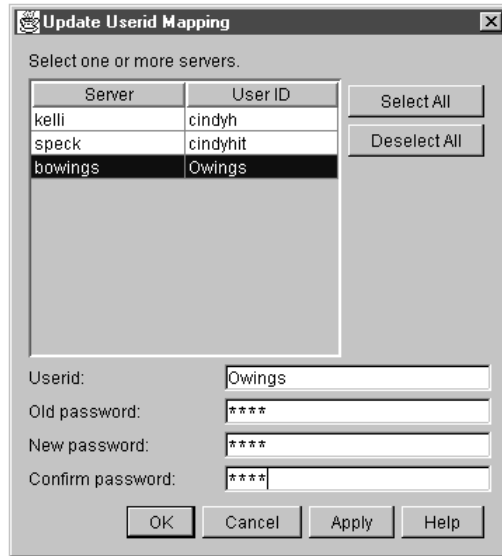


Figure 17. Update Userid Mapping window

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**

For servers that support search templates, this bean displays a list of available search templates and allows selection of a template.

The CMBSearchTemplateList bean has three styles. The image style, shown in Figure 18 on page 309, uses one image for the backgrounds of the selected items and another for the unselected items. Figure 19 on page 309 shows the simple template list style. Figure 20 on page 309 shows the drop-down template list style.

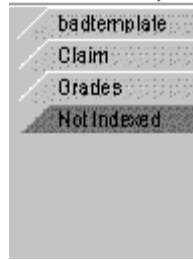


Figure 18. Image template list style



Figure 19. Simple template list style



Figure 20. Drop-down template list style

## CMBSearchTemplateViewer bean

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.

The CMBSearchTemplateViewer bean (see Figure 21) 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 21. 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.

Provide validation logic 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

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.

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.

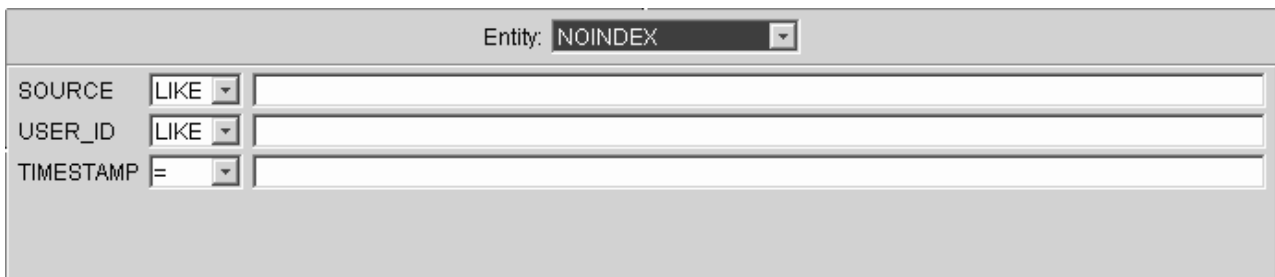


Figure 22. `CMBSearchPanel` Bean

## `CMBSearchResultsViewer` bean

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.

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 23 shows the CMBSearchResultsViewer bean with the **Search Results** folder selected.

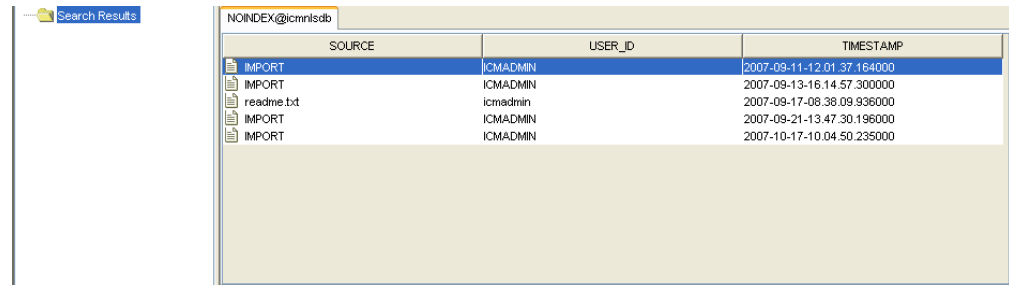


Figure 23. CMBSearchResultsViewer bean

### 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 entity (item type) 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

Displays the contents of one or more folders in a Windows Explorer style window.

The `CMBFolderViewer` bean displays a tree pane that looks like the `CMBSearchResultsViewer` bean. There is no main **Search Results** folder. Figure 24 shows the tree and details panes of the `CMBFolderViewer` bean.

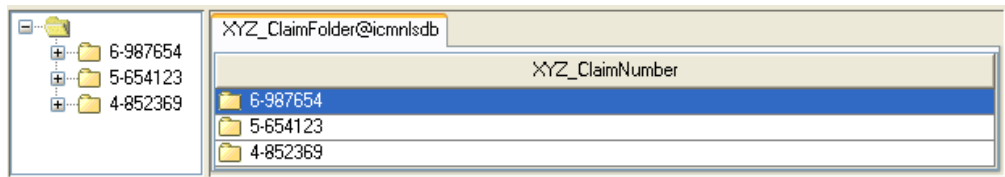


Figure 24. `CMBFolderViewer` bean

The `CMBFolderViewer` bean displays a tree of folders on the left pane. The right pane displays a notebook of tables of the documents contained by folder selected in the tree pane. A resizable splitter separates the tree and notebook panes.

#### CMBFolderViewer Tree pane

The tree pane contains folders. Nested folders appear beneath each folder.

#### CMBFolderViewer Details pane

The details pane contains the contents of the folder that is selected in the tree pane. The contents display in a notebook with a tab for each 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**

Displays one or more documents using the Java viewer or by launching a separate viewer application.

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:

### **Java-based viewers**

These viewers must extend the class CMBJavaDocumentViewer.

### **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.

**Traffic Collision Report**

Report Number: 7292001-73  
 Location of Collision: Southbound Highway 405  
 Date: 07/29/2001 Time: 16:37

Page 1 of 1

| Party 1 | First Name | Last Name | Driver's License Number | State |
|---------|------------|-----------|-------------------------|-------|
|         | George     | Tolstoy   | B25878914               | CA    |

Address: 972 Willow St.  
 City: Marina State: CA Zip Code: 90548  
 Home Phone: (695) 310-1395 Work Phone: \_\_\_\_\_

| Sex | Height | Weight  | Hair | Birth Date | Insurance Company | Policy Number |
|-----|--------|---------|------|------------|-------------------|---------------|
| M   | 5' 4"  | 110 Lbs | blk  | 06/27/63   | First Insurance   | 33746119237   |

| VEH 1 | Make  | Model   | Year | Color | License Plate | State |
|-------|-------|---------|------|-------|---------------|-------|
|       | Acura | Integra | 1990 | Blue  | 6USA676       | CA    |

Shade Vehicle Damage Areas

| Party 2 | First Name | Last Name | Driver's License Number | State |
|---------|------------|-----------|-------------------------|-------|
|         | Gabriela   | Neruda    | B22451308               | CA    |

Figure 25. CMBDocumentViewer Bean

## Viewer specifications

Viewers must be specified before building an application.

There are two ways to specify viewers:

- 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.
- 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 DB2 Information Integrator for Content Administration and using theMime2App 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

You can launch external viewers by using the Mime2App property.

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

Displays a window where users can update the index class and indexing attributes for an item.

The CMBItemAttributesEditor bean (see Figure 26) displays a window for viewing and modifying the entity (item type) and attributes of a folder or document.

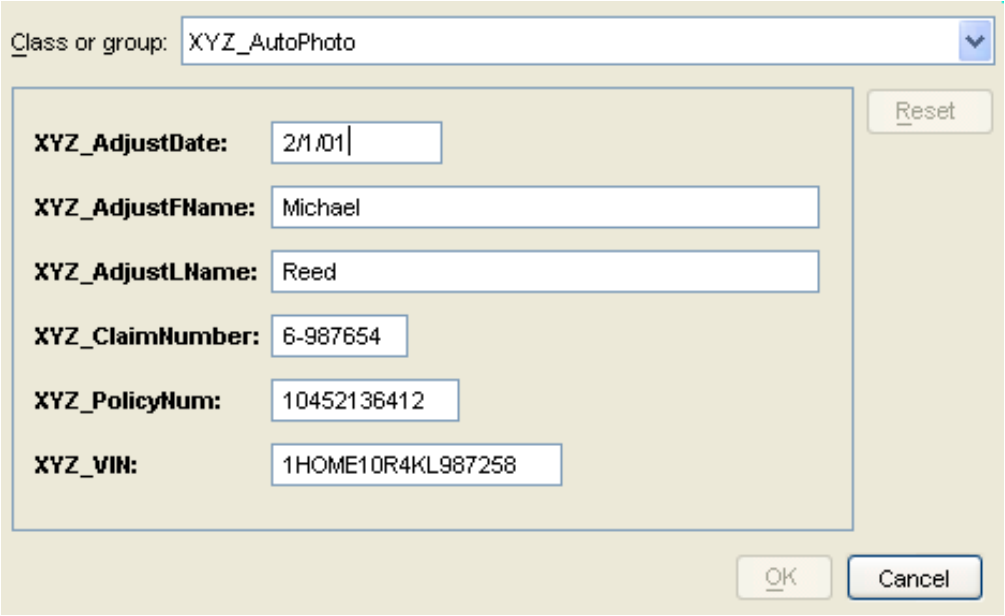


Figure 26. CMBItemAttributesEditor bean

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.

The validation logic to veto changes in the CMBItemAttributesEditor can be provided by a handler for the CMBEditRequestedEvent.

## CMBVersionsViewer bean

Displays version information for a document, if versioning is enabled.

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.

| DraftReport@BOWINGS2                                                                  |                        |            |              |              |                |
|---------------------------------------------------------------------------------------|------------------------|------------|--------------|--------------|----------------|
| 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 27. 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.

Visual beans have the following three properties:

### 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

To save and restore fields, use the loadConfiguration and saveConfiguration methods.

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) of 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 replace a visual bean with another bean or a Swing component, 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 42.

Table 42. Visual beans and key events

| Visual bean             | Key events                                                         |
|-------------------------|--------------------------------------------------------------------|
| CMBSearchTemplateList   | CMBTemplateSelectedEvent                                           |
| CMBSearchTemplateViewer | CMBSearchStartedEventCMBSearchResultsEvent                         |
| 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 CMBCConnection non-visual bean.

## Building an application using visual beans

The following sections show how the visual beans fit together when you build an application.

A sample client application that was written using the visual beans is provided for you. The source files for the sample are located in: %IBMCMROOT%/samples/java/beans/gui.. You should also read the readme.html file in this directory for details about the sample client and setup requirements.

### 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 28 on page 319, 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.

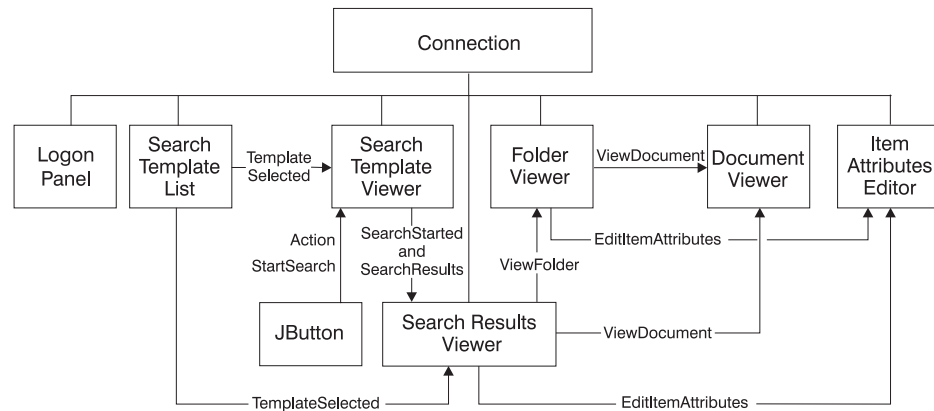


Figure 28. Visual bean connections

Figure 28 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. Enter a user ID and password into the logon window and clicks **OK**. The CMBLogonPanel bean invokes the connect method of the CMBCConnection bean to establish the connection to the server. 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 CMBCConnection bean. CMBSearchTemplateList sets up the listeners when a CMBCConnection bean associated itself with it using the setConnection method.)
2. Select 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.
3. Complete the template, and either press **Enter** or click **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. 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. 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.
4. Use CMBSearchCompleted event to enable the **Search** button again if it was disabled at the start of the search.
5. Expand folders in the Search Results window, then select a document or folder for viewing. A CMBViewFolderEvent or CMBViewDocumentEvent is generated. The CMBFolderViewer and CMBDocumentViewer beans are listening to their respective events, and display the folder or document.
6. From the CMBFolderViewer, 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.
7. To update, select a document's or folder's attributes from the CMBSearchResultsViewer or CMBFolderViewer. Selecting a document generates

a `CMBEditItemAttributesEvent`. The `CMBItemAttributesEditor` bean listens for an `CMBEditItemAttributesEvent`. It displays the entity and attributes for the item.

8. Change the entity and attributes and then click **OK** to apply the changes.

### Using beans in more than one window or dialog

You must provide additional code to pass an event from a bean in one window to a bean in another window.

Typically, the fact that an event has been sent is usually the reason for displaying a window. The `EditAttributesDialog` window contains the `ItemAttributesEditor`. `SearchFrame` creates the window when a `CMBEditItemAttributesEvent` launches:

```
// Invoke a secondary dialog for edit attributes
searchResultsViewer.addEditItemAttributesListener(new
CMBEditItemAttributesListener() {
 public void onEditItemAttributes(CMBEditItemAttributesEvent event) {
 EditAttributesDialog editAttributesDialog = new
 EditAttributesDialog(SearchFrame.this,connection,event.getItem());
 editAttributesDialog.setVisible(true);
 }
});
```

The information that is normally passed to the `CMBItemAttributesEditor` bean is passed as arguments to the constructor of the window instead. Within the constructor, the information is passed to the `CMBItemAttributesEditor` bean by setting the following properties:

```
itemAttributesEditor.setConnection(connection);
```

```
itemAttributesEditor.setItem(item);
```

---

## Working with XML services (Java only)

The DB2 Content Manager connector provides APIs in `cmbxmlservice.jar` that can convert three types of system administration data into XML:

- Administrative metadata objects, which can include users, user groups, resource manager configuration, document routing definitions, ACL privileges, privilege sets, library server configuration, and the DB2 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 DB2 Information Integrator for Content entities. These objects are stored as XSD files (storage schemas), which 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.

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.

---

## 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 run-time 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.

### 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 run-time 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.

**Schema mapping utility (XML)**

XML conversion tool that can convert a user-defined schema into the storage schema that DB2 Content Manager supports.

**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 29 illustrates how these XML layers relate to the DB2 Content Manager connector.

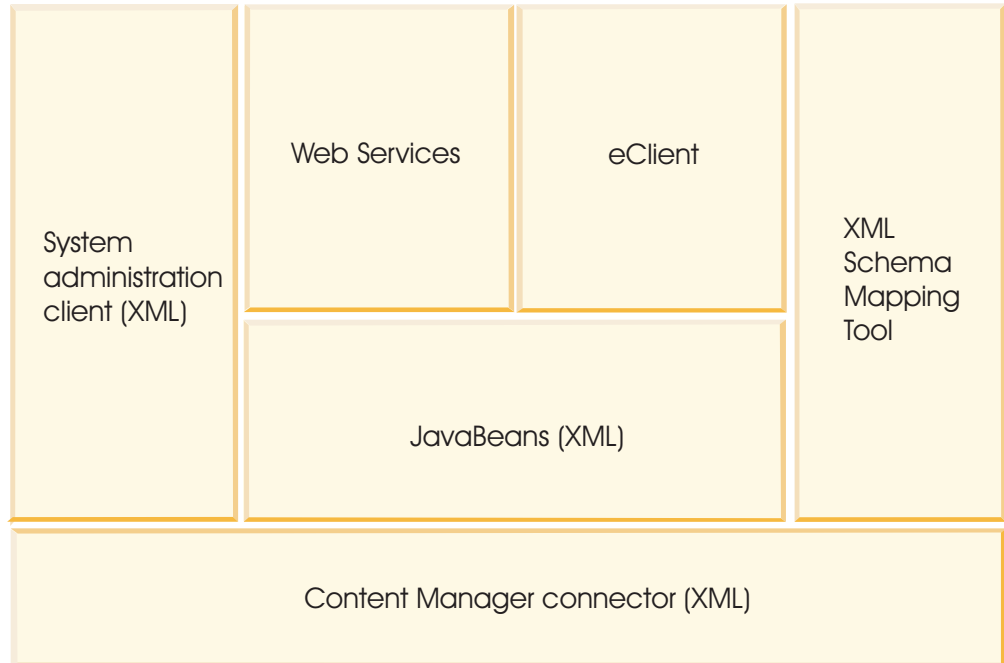


*Figure 29. XML services programming layers*

Figure 30 on page 323 depicts a real-world scenario for communicating with DB2 Content Manager through its XML interface:

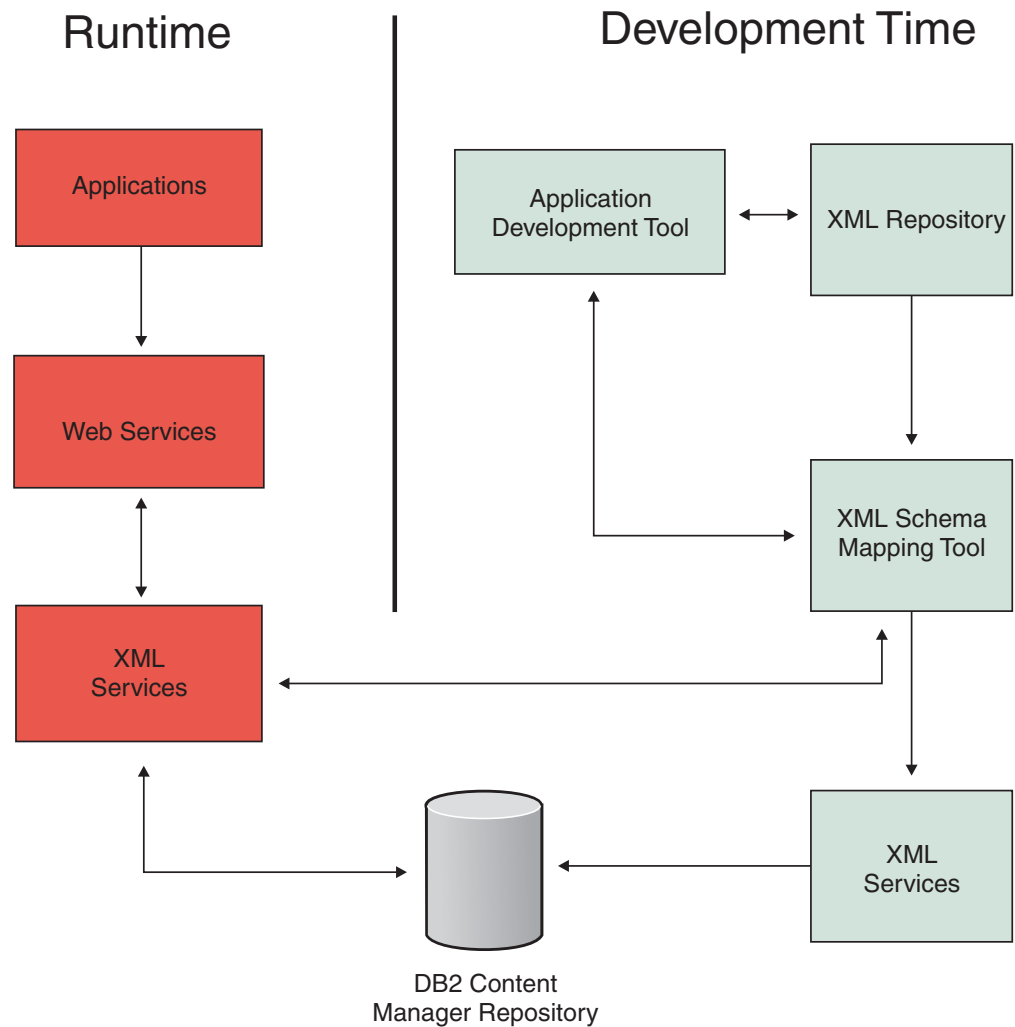


Figure 30. XML services communication flowchart

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, for example, 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 services 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.

#### Related information





Programming run-time operations through XML JavaBeans



Mapping a user-defined schema to a storage schema with the XML schema mapping tool

---

## 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.

The types of metadata that you can import and export by using the DB2 Content Manager APIs are:

### Administrative objects

Users, user groups, resource manager configuration, document routing definitions, ACL privileges, and the DB2 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

The structure of item types, component types, and DB2 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 DB2 Information Integrator for Content system to another DB2 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 many 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.

### **Related information**



Importing and exporting XML object dependencies

## Importing and exporting administration objects as XML

The XML instance service class, DKXMLSysAdminService, contains two 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 that creates it. That is, by exporting and importing an object from one system to another system, all of 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.

**Restriction:** XML services on z/OS does not support import or export of resource manager objects.

## XML example

The following example creates a new user (Joshua) and a new group (XMLDev) in the DB2 Content Manager server:

```
<?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">
 <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 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">`
- In accordance with the SQL/XML standard for mapping SQL identifiers to XML names, the XML schema converter automatically escapes special characters with the unicode equivalent (in the form of `_xYYYY`, where `YYYY` represents the Unicode string). For example:
  - Elements and attribute names cannot start with the letters XML. Therefore, if an item type is named `XMLDocument`, then its new name becomes: `_x0058_MLDocument`.
  - Elements and attributes names cannot contain spaces. Therefore, if a federated entity is named `Project Entity`, then its new name becomes: `Project_x0020_Entity`.

For a complete list of valid characters in XML, see the XML standard at <http://www.w3.org/TR/REC-xml#NT-Name>.

The exported schema for a given data model object is semantically equivalent to the imported schema that creates it. That is, by exporting and importing an object from one system to another system, all of the imported object's properties remain the same as those of the original exported one. However, the exported schema document and the imported schema document might differ in syntax 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 43 on page 328.
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 44 on page 334.
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. These are shown in Table 44 on page 334.

Table 43. 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: <attribute name=".." type=".." /> 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: <attribute name="XX" type=".." />                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                |
| -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:                     |           |          | This represents the derived string simple type definition with the minLength and maxLength constraints. For example: <simpleType name="char_100"> <restriction base="string"> <length value="100" /> </restriction> </simpleType> Note that minLength and maxLength must be the same. There is one simple type definition per each attribute type instance with different length. The simpleType is tied to the attribute declaration in the export file. During import, the simpleType 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. | The length can be specified.                                   |
| --alphabetic                     |           |          | You can use the pattern element in the restriction to enforce the type. For example: <simpleType name="alpha_char_100"> <restriction base='string'> <pattern value="[a-zA-Z]*" /> <length value="100" /> </restriction> </simpleType>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                |
| --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                        |           |          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                |

Table 43. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

| Object                  | Data type | Required | Corresponding storage schema construct                                                                                                                                                                                                                                                                    | Comments                                    |
|-------------------------|-----------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| --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:<br><attribute name=".." type="base64Binary"><br><annotation><appinfo> <CM:dataType value="BLOB"/> </appinfo> </annotation><br></attribute>                                                                                  | The length can be specified.                |
| -CLOB                   |           |          | This maps to a regular string simple type with a CM meta-attribute namespace. For example:<br><attribute name="..." type="string"><br><annotation><appinfo> <CM:datatype value="CLOB"/> </appinfo></annotation><br></attribute>                                                                           | The length can be specified.                |
| -max                    | integer   |          | The maxInclusive element in the derived simple type definition. For example:<br><simpleType name='integer_max100'><br><restriction base='integer'> <maxInclusive value="100"/> </restriction> </simpleType><br>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:<br><simpleType name='integer_min10'><br><restriction base='integer'> <minInclusive value="10"/> </restriction> </simpleType>                                                                                                 | This is only for short and integer.         |
| -length                 | integer   |          | The length element in the derived simple type definition. For example: <simpleType name='char_256'> <restriction base="string"><br><length value="256"/> </restriction><br></simpleType>                                                                                                                  |                                             |

Table 43. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

| Object                    | Data type       | Required | Corresponding storage schema construct                                                                                                                                                                                                                                                                                                                                                                                                                                              | Comments                                                                                                                                     |
|---------------------------|-----------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| -scale/precision          | integer         |          | The fractionDigits and totalDigits elements in the derived simple type definition. For example: <code>&lt;simpleType name='decimal_8,3'&gt;<br/>&lt;restriction base='decimal'&gt; &lt;fractionDigits value='3' /&gt; &lt;totalDigits value='8' /&gt;<br/>&lt;/restriction&gt; &lt;/simpleType&gt;</code>                                                                                                                                                                           | This is only for decimal.                                                                                                                    |
| Attribute groups (global) |                 |          | The attributeGroup element in the XSD. For example: <code>&lt;xs:attributeGroup name='myAttrGrp'&gt; &lt;xs:attribute .../&gt; ...<br/>&lt;/xs:attributeGroup&gt;</code>                                                                                                                                                                                                                                                                                                            |                                                                                                                                              |
| -Name                     | string          | Yes      | The name attribute in the attributeGroup element. For example: <code>&lt;xs:attributeGroup name='myAttrGrp'&gt; &lt;xs:attribute .../&gt; ...<br/>&lt;/xs:attributeGroup&gt;</code>                                                                                                                                                                                                                                                                                                 |                                                                                                                                              |
| -Attributes               | array of string | yes      | The child elements of the attributeGroup element. For example, <code>&lt;xs:attributeGroup name='myAttrGrp'&gt; &lt;xs:attribute .../&gt; ...<br/>&lt;/xs:attributeGroup&gt;</code> 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, <code>myAttrGrp.myStringAttr</code> . | The is the array of attribute names.                                                                                                         |
| Reference attribute       |                 |          | This maps to a string attribute with a specific XML schema annotation. 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.                                                                                                                                     |                                                                                                                                              |
| -Name                     | string          | yes      | Maps to the attribute name.                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                              |
| Item Type: definition     |                 |          | A global element definition, with <code>xs:schema</code> element as the parent. For example: <code>&lt;xs:schema ....&gt; &lt;xs:element name='elem1'&gt; ....<br/>&lt;/xs:element&gt; &lt;/xs:schema&gt;</code> 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.                                                                                                |                                                                                                                                              |
| -Name                     | string          | yes      | The name attribute in the <code>xs:element</code> declaration. For example: <code>&lt;xs:schema ....&gt; &lt;xs:element name='elem1'&gt; ....<br/>&lt;/xs:element&gt; &lt;/xs:schema&gt;</code>                                                                                                                                                                                                                                                                                     |                                                                                                                                              |
| -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 <code>cmdatamodel.xsd</code> schema file. For example: <code>&lt;element ref='cm:properties' minOccurs='0'<br/>maxOccurs='unbounded'&gt;</code>                                                                                                                                                                                | Use in the item instance to describe any instance-level properties associated with the item, such as semantic type, ACL, creation time, etc. |



Table 43. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

| Object                                                                                     | Data type    | Required | Corresponding storage schema construct                                                                                                                                                                                                                                                                                                                                                                         | Comments                                                                                                                                |
|--------------------------------------------------------------------------------------------|--------------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| -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.                                                     |
| -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<br>• Non-resource item<br>• Resource item<br>• Document item type | short        |          |                                                                                                                                                                                                                                                                                                                                                                                                                | Represents the item, resource item, document, or document part. Use the ItemTypeRelation DefICM to represent relationships to the part. |
| -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.                                                                                                                                                                                                                                      |                                                                                                                                         |



Table 43. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

| Object                | Data type | Required | Corresponding storage schema construct                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Comments |
|-----------------------|-----------|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| 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. |          |
| -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:<br><xsd:attribute name=".." use="required" .../><br>Here is how you map the value of nullable attribute to the "use" and "default" attribute.<br>"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:<br><xsd:attribute name=".." default ="10" ... />                                                                                                                                                                                                                                                                                                                                                                                                                              |          |

Table 43. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

| Object                                                                                                                 | Data type | Required | Corresponding storage schema construct                                                                                                                                                                                                                                                               | Comments                                 |
|------------------------------------------------------------------------------------------------------------------------|-----------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------|
| Sub-components /<br>child components                                                                                   |           |          | This maps to a child element declaration (using sequence) with the min and max occurrence of a child component mapped to the minOccurs and maxOccurs of the element declaration.                                                                                                                     |                                          |
| 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 ....><br><xs:element name="elem1"><br><xs:annotation><xs:appinfo> <xs:entityType value="federated"/> </xs:appinfo></xs:annotation> .... </xs:element><br></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 ....><br><xs:element name="elem1"><br><xs:annotation><xs:appinfo> <xs:entityType value="federated"/> </xs:appinfo></xs:annotation> <xs:complexType> <xs:attribute name="attr1" ....> .... </xs:element><br></xs:schema>  |                                          |
| --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).                                                                                                                                                           |                                          |

Table 43. How DB2 Content Manager data model objects map to the storage schema constructs (continued)

| Object              | Data type | Required | Corresponding storage schema construct | Comments |
|---------------------|-----------|----------|----------------------------------------|----------|
| -Fed entity name    | string    |          |                                        |          |
| -Fed attr name      | string    |          |                                        |          |
| -Native server name | string    |          |                                        |          |
| -Native server type | string    |          |                                        |          |
| -Native entity name | string    |          |                                        |          |

For Table 44, 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 44. 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 <code>cm:stringType</code> 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, <code>&lt;cm:stringType value="ALPHA" /&gt;</code> | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --numeric                        | <code>&lt;cm:stringType value="NUMERIC" /&gt;</code>                                                                                                                                                                                                                                      | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --alphanumeric                   | <code>&lt;cm:stringType value="ALPHANUMERIC" /&gt;</code>                                                                                                                                                                                                                                 | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --Extended alphanumeric          | <code>&lt;cm:stringType value="ALPHANUMERIC_EXT" /&gt;</code>                                                                                                                                                                                                                             | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --other                          | <code>&lt;cm:stringType value="OTHER" /&gt;</code>                                                                                                                                                                                                                                        | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| -Variable character              |                                                                                                                                                                                                                                                                                           |                           |                            |

Table 44. 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 |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------|---------------------------|----------------------------|
| --alphabetic                 | Same as in the Character case<br><cm:stringType value="ALPHA"/>                                                           | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --numeric                    | <cm:stringType<br>value="NUMERIC"/>                                                                                       | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --alphanumeric               | <cm:stringType<br>value="ALPHANUMERIC"/>                                                                                  | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --Extended alphanumeric      | <cm:stringType<br>value="ALPHANUMERIC_EXT"/>                                                                              | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| --other                      | <cm:stringType value="OTHER"/>                                                                                            | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| -Short integer               |                                                                                                                           |                           |                            |
| -Long integer                |                                                                                                                           |                           |                            |
| -Double                      |                                                                                                                           |                           |                            |
| -Decimal                     |                                                                                                                           |                           |                            |
| -Date                        |                                                                                                                           |                           |                            |
| -Time                        |                                                                                                                           |                           |                            |
| -Timestamp                   |                                                                                                                           |                           |                            |
| -BLOB                        | <CM:dataType value="BLOB"/>                                                                                               | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| -CLOB                        | <CM:dataType value="CLOB"/>                                                                                               | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| -max                         |                                                                                                                           |                           |                            |
| -min                         |                                                                                                                           |                           |                            |
| -length                      |                                                                                                                           |                           |                            |
| -scale/precision             |                                                                                                                           |                           |                            |
| -Description w/<br>lang code | cm:description element with a value<br>attribute and xsi:lang attribute (to<br>indicate which language it belongs<br>to). | GLOBAL                    |                            |
| Attribute groups<br>(global) |                                                                                                                           |                           |                            |
| -Name                        |                                                                                                                           |                           |                            |
| -Attributes                  |                                                                                                                           |                           |                            |
| -Description w/<br>lang code | cm:description element with a value<br>attribute and xsi:lang attribute (to<br>indicate which language it belongs<br>to). | GLOBAL                    |                            |

Table 44. 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                                            |
|----------------------------|------------------------------------------------------------------------------------------------------------------|---------------------------|-----------------------------------------------------------------------|
| Reference attribute        | Indicated by an annotation element, cm:referenceAttribute with the following attributes:                         | GLOBAL,<br>LOCAL,<br>VIEW |                                                                       |
| -Name                      |                                                                                                                  |                           |                                                                       |
| -Reference delete rule     | {cm:referenceAttribute}<br>cm:deleteRule=<string> in enumeration type {NO_ACTION, SET_NULL, CASCADE, RESTRICT}.  | GLOBAL,<br>LOCAL,<br>VIEW |                                                                       |
| -Reference sequence number | {cm:referenceAttribute}<br>cm:sequenceNumber=<short>                                                             | GLOBAL,<br>LOCAL,<br>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 44. 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                           |                                                                                                                                                                                                                                                        |                         | The cmdatamodel.xsd file contains the definition of the links, which is <element name="links"> <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> Note that this will be part of the pre-defined DB2 Content Manager schema file, cmdatamodel.xsd, to be imported. |
| 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                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| -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                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

Table 44. 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                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -Prefetch coll name                                     | {cm:resourceItemInfo}<br>cm:prefetchCollection element with name and attributes.                                                                                             | GLOBAL                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| -an embedded binary object (used in the instance level) |                                                                                                                                                                              |                         | The cmdatamodel.xsd contains the definition of the resourceObject, which is<br><element name="cm:resourceObject"><br><complexType> <choice> <element name="content"> <complexType><br><attribute name="value" type="base64Binary"/> </complexType><br></element> <element name="url"><br><complexType> <attribute name="value" type="anyURI"/> </complexType><br></element> </complexType> </element><br>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. |
| 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                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| -ACL name                                               | cm:ACL element with name and attributes under the referenced element. For example: <element ref="cm:ICMBASE" ...><br><xsd:annotation> <xsd:appinfo><br><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                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

Table 44. 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 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}<br>sourceItemtype=<string>                                                                                                        | GLOBAL                  |                            |
| -Item type to be linked     | {cm:autoLinkAttributes}<br>cm:targetItemtype=<string>                                                                                                     | GLOBAL                  |                            |
| -Auto linking attribute     | {cm:autoLinkAttributes}<br>cm:attributeName=<string>                                                                                                      | GLOBAL                  |                            |
| -Auto linking attribute     | {cm:autoLinkAttributes}<br>cm:attributeGroupName=<string>                                                                                                 | GLOBAL                  |                            |
| -Link type                  | {cm:autoLinkAttributes}<br>cm:autoLinkType=<string>                                                                                                       | GLOBAL                  |                            |
| -Auto Linking SMS           | A cm:autoLinkingSMSRule element with a value attribute.                                                                                                   | GLOBAL                  |                            |
| logging                     |                                                                                                                                                           |                         |                            |
| -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}<br>cm:constraintName=<string>                                                                                                             | GLOBAL                  |                            |
| -Update rule                | {cm:foreignKey}<br>cm:updateRule={RESTRICT, NO_ACTION}                                                                                                    | GLOBAL                  |                            |



Table 44. 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 |
|-----------------------------|-------------------------------------------------------------------------------------------------|-------------------------|----------------------------|
| -Delete rule                | {cm:foreignKey}<br>cm:deleteRule={RESTRICT,<br>CASCADE, NO_ACTION,<br>SET_NULL}                 | GLOBAL                  |                            |
| -Source component           | {cm:foreignKey}<br>cm:sourceComponent=<string>                                                  | GLOBAL                  |                            |
| -Target item type           | {cm:foreignKey}<br>cm:targetItemType=<string>                                                   | GLOBAL                  |                            |
| -Target external table      | {cm:foreignKey}<br>cm:targetTable=<string>                                                      | GLOBAL                  |                            |
| -Attribute Pairs            | A cm:attributePair element under the<br>cm:foreignKey element with the<br>following attributes: | GLOBAL                  |                            |
| -Source attribute group     | {cm:attributePair}<br>cm:sourceAttributeGroup=<string>                                          | GLOBAL                  |                            |
| -Source attribute           | {cm:attributePair}<br>cm:sourceAttribute=<string>                                               | GLOBAL                  |                            |
| -Target attribute group     | {cm:attributePair}<br>cm:targetAttributeGroup=<string>                                          | GLOBAL                  |                            |
| -Target attribute           | {cm:attributePair}<br>cm:targetAttribute=<string>                                               | GLOBAL                  |                            |
| -External table column name | {cm:attributePair}<br>cm:targetTableColumn=<string>                                             | GLOBAL                  |                            |
| User exits                  | cm:userExit element with the<br>following attributes:                                           | GLOBAL,<br>VIEW         |                            |
| -Exit name                  | {cm:userExit} cm:name=<string>                                                                  | GLOBAL,<br>VIEW         |                            |
| -Function name              | {cm:userExit}<br>cm:functionName=<string>                                                       | GLOBAL,<br>VIEW         |                            |
| -DLL name                   | {cm:userExit} cm:DLLName=<string>                                                               | GLOBAL,<br>VIEW         |                            |
| Access Module               | A cm:accessModule element with the<br>following attributes:                                     | GLOBAL                  |                            |
| -name                       | {cm:accessModule}<br>cm:name=<string>                                                           | GLOBAL                  |                            |
| -result                     | {cm:accessModule}<br>cm:result=<integer>                                                        | GLOBAL                  |                            |
| -status                     | {cm:accessModule}<br>cm:status=<short>                                                          | GLOBAL                  |                            |
| -version                    | {cm:accessModule}<br>cm:version=<short>                                                         | GLOBAL                  |                            |
| Previous Access Module      | A cm:previousAccessModule element<br>with a version attribute.                                  | GLOBAL                  |                            |
| Component Type:             |                                                                                                 |                         |                            |
| -Attribute groups           |                                                                                                 |                         |                            |
| -Reference attributes       |                                                                                                 |                         |                            |

Table 44. 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                 |                                                                          |                           |                            |
| -Text Searchable            | A cm:textSearchable element, with a value attribute in boolean type.     | GLOBAL,<br>LOCAL,<br>VIEW |                            |
| -Text Index Information     | A cm:textIndexInfo element, with the following attributes:.              | GLOBAL,<br>LOCAL          |                            |
| -Commit count               | {cm:textIndexInfo }<br>cm:commitCount=<integer>                          | GLOBAL,<br>LOCAL          |                            |
| -Format                     | {cm:textIndexInfo }<br>cm:format=<integer>                               | GLOBAL,<br>LOCAL          |                            |
| -Index CCSID                | {cm:textIndexInfo }<br>cm:CCSID=<integer>                                | GLOBAL,<br>LOCAL          |                            |
| -Index Directory            | {cm:textIndexInfo }<br>cm:directory=<string>                             | GLOBAL,<br>LOCAL          |                            |
| -Index Language Code        | {cm:textIndexInfo }<br>cm:langCode=<string>                              | GLOBAL,<br>LOCAL          |                            |
| -Minimum changes            | {cm:textIndexInfo }<br>cm:minChanges=<integer>                           | GLOBAL,<br>LOCAL          |                            |
| -Model File                 | {cm:textIndexInfo }<br>cm:modelFile=<string>                             | GLOBAL,<br>LOCAL          |                            |
| -Model Name                 | {cm:textIndexInfo }<br>cm:modelName=<string>                             | GLOBAL,<br>LOCAL          |                            |
| -Model CCSID                | {cm:textIndexInfo }<br>cm:modelCCSID=<integer>                           | GLOBAL,<br>LOCAL          |                            |
| -UDF Name                   | {cm:textIndexInfo }<br>cm:UDFName=<string>                               | GLOBAL,<br>LOCAL          |                            |
| -UDF Schema                 | {cm:textIndexInfo }<br>cm:UDFSchema=<integer>                            | GLOBAL,<br>LOCAL          |                            |
| -Update Frequency           | {cm:textIndexInfo }<br>cm:updateFrequency=<string>                       | GLOBAL,<br>LOCAL          |                            |
| -Update Frequency Unit      | {cm:textIndexInfo }<br>cm:updateFrequencyUnit=<{MINUTE, HOUR}>           | GLOBAL,<br>LOCAL          |                            |
| -Working Directory          | {cm:textIndexInfo }<br>cm:workingDir=<string>                            | GLOBAL,<br>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                     |                            |

Table 44. 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 |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|----------------------------|
| -Database indexes                   | A cm:databaseIndexInfo element, with the following attributes:                                                                                                                                     | GLOBAL,<br>LOCAL        |                            |
| --Name                              | {cm:databaseIndexInfo }<br>cm:name=<string>                                                                                                                                                        | GLOBAL,<br>LOCAL        |                            |
| --Unique                            | {cm:databaseIndexInfo }<br>cm:unique=<boolean>                                                                                                                                                     | GLOBAL,<br>LOCAL        |                            |
| --Index Schema                      | {cm:databaseIndexInfo }<br>cm:indexSchema=<string>                                                                                                                                                 | GLOBAL,<br>LOCAL        |                            |
| --Attributes                        | A subelement, cm:indexedAttribute element under the same cm:databaseIndexInfo element with the following attributes:                                                                               | GLOBAL,<br>LOCAL        |                            |
| ---Name                             | {cm:indexAttribute }<br>cm:name=<string>                                                                                                                                                           | GLOBAL,<br>LOCAL        |                            |
| ---Index Order                      | {cm:indexAttribute }<br>cm:indexOrder={ASCENDING,<br>DESCENDING}                                                                                                                                   | GLOBAL,<br>LOCAL        |                            |
| Delete rule                         | A cm:deleteRule element with a value attribute {RESTRICT,<br>CASCADE}.                                                                                                                             | LOCAL                   |                            |
| Sub-components/<br>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                    |                            |
| 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:writable 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                    |                            |

Table 44. 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 |
|------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|-------------------------|----------------------------|
| --View<br>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                  |                            |
| -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                  |                            |

## Example: XML schema

The following example shows a sample storage schema (XSD) snippet for the XYZ Insurance policy item type:

```
<?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>

```

## Unsupported XML types in the DB2 Content Manager storage schemas

The DB2 Content Manager storage schemas do not support the following primitive data types:

- 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 data types:

- normalizedString
- token
- language
- NMTOKEN
- NMTOKENS

- Name
- NCName
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- nonPositiveInteger
- negativeInteger
- long
- byte
- nonNegativeInteger
- unsignedLong
- unsignedInt
- unsignedShort
- unsignedByte
- positiveInteger

### Constraints for converting to DB2 Content Manager storage schemas

The following constraints also apply to the DB2 Content Manager storage schema conversion:

- 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 methods for importing and exporting XML items: ingest() and extract().

These ingest() and extract() methods take the XML file that conforms to the storage schema (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 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.

### Example: XML item instance

The following example shows a sample item instance that conforms to the XYZ Insurance policy storage schema:

```
<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>
```

#### Related information

 Importing and exporting DB2 Content Manager data model objects as XML schema files (XSD)

## Exporting DB2 Content Manager DDO items as XML items

The extract() method in DKXMLDataService 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.

**Important:** 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. Use this option to exclude all proprietary DB2 Content Manager information from the XML item.

## Example: Java

The following example inputs a ddo item and returns it as an xmlobj XML document (in DOM format with the PID embedded in it)n returns system and resource properties in a separate file, and returns resource content as an input stream.

```
//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 DB2 Content Manager server. These constructors extract content



from an XML document, and 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 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 (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 that describes the item. You can either provide this document through the setItemProperties() method or in the constructor.

## Example: Java

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.

```
//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()
```

### Related information




---

## Importing and exporting XML object dependencies

Scenarios can occur where data model and administrative 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 only data model dependency objects.

### **DK\_CM\_XML\_EXPORT\_SA\_ONLY\_PREREQUISITE**

Exports only administrative dependency objects (including authorization, authentication, and library server configuration).

### **DK\_CM\_XML\_EXPORT\_RM\_ONLY\_PREREQUISITE**

Exports only resource manager configuration (in the library server side) dependency objects.

### **DK\_CM\_XML\_EXPORT\_DR\_ONLY\_PREREQUISITE**

Exports only document routing dependency objects.

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.

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 the first step for using the APIs, you 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.

As the second step in using the APIs, you 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.

## XML services support on z/OS

The XML schema mapping tool APIs are not supported natively on z/OS, which means that you cannot run the APIs, which includes classes `DKSchemaConverter`, `DKDocumentConverter`, and `DKMapperException` (or any `com.ibm.mm.sdk.xml.schema.*` classes) on z/OS for the schema transformation. However, the transformation might be performed using the XML schema mapping tool on a non-z/OS platform (such as Windows) first and the resultant XSD or XML file can be imported into the DB2 Content Manager on z/OS. The import can be performed in one of three ways:

- Using the XML Schema mapping tool GUI interface directly (by connecting to the DB2 Content Manager server on z/OS).
- The transformed XSD/XML file can be downloaded onto z/OS and imported using the native XML Services API on z/OS.
- Using the XML APIs on a non-z/OS platform to import the transformed XSD/XML file into DB2 Content Manager on z/OS.

## Example: Converting XML 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]);

```

## Example: Converting XML documents

### 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");

```

### Related information



Mapping and importing XML schemas

---

## Programming run-time 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 run-time 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 CMBXMLServices bean (similar to what the Web services do). Your XML requests must follow the structure described in the cmbmessages.xsd schema.

When making requests to the XML Java beans, you might have to specify the URI attribute for an item or folder that you want to work with. The URI is a unique resource identifier, and it may take one of two forms. The PID of the item as it is stored in DB2 Content Manager is a valid URI which uniquely identifies the item. You specify the PID as the URI for any items or folders in all requests to the web services and XML beans. When you get replies from the Eeb services, the URI will be returned in the form of a URL.

```
http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
92 3 ICM8 icmnlbdb13 XYZ_InsPolicy59 26 A1001001A04I10B45558I8302118
93 A04I10B45558I830211 14 1029&server=icmnlbdb&dsType=ICM
```

You can also use the URL in the previous example as the value for the URI attribute in all requests to the web services or to the XML beans.

The following JavaBean example sets up a CMBXMLMessage bean to send an XML search request directly to it:

### Example: Java

```
public class TXMLSearch2 {
 public static void main(String[] args) throws Exception {
 String dstype = args[0];
 String server = args[1];
 String userid = args[2];
 String password = args[3];
 String entity = args[4];
 String condition = "";
 // 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. ICMNLDB(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:

### Example: 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 +
\" xmlns='\" + CMBXMLConstant.namespaceURI +
\" xmlns:cm='\" + CMBXMLConstant.cmNamespaceURI + \">");
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());
if (reply.getAttachments() != null) System.out.
 println("There are " + reply.getAttachments().length + " attachments");
return reply;
}
}

```

## Listing DB2 Content Manager servers with ListServerRequest

Create a <ListServerRequest> to list all available DB2 Content Manager servers.

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 servers known to the system.

### Example: XML request

The following example lists all servers of type ICM:

```
<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.

### Example: XML reply

```
<ListServersReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<ServerDef>
 <ServerType>ICM</ServerType>
 <ServerName>icm1sdb</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).

### <LoginData> (required)

Authenticates the user by either user ID/password or by a WebSphere SSO credential.

## <AuthenticationData> attributes

### connectString (optional)

Passes a server-specific property (beyond user ID and password) to establish a connection to the server. When sending requests to the CMBXMLServices bean directly, always specify SCHEMA=*userid* where *userid* is the schema of the DB2 Content Manager V8 database that you are connecting to. Typically this is ICMADMIN.

### configString (optional)

Passes a CMBConnection property value to help construct a dkDatastore instance.

### connectToWorkflow (optional)

Toggles an option to connect to the workflow server. The default is TRUE.

### Related information



Listing DB2 Content Manager servers with ListServerRequest

## Changing a password with XML requests

<ChangePasswordRequest> changes your password in DB2 Content Manager.

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.

### <NewPassword> (required)

Specifies the new password to replace the old one in <AuthenticationData>.

## Example: XML request

The following example changes the password of user ID testuser to passw0rd2:

```
<ChangePasswordRequest
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>testuser</UserID><Password>passw0rd</Password></LoginData>
 </AuthenticationData>
 <NewPassword>passw0rd2</NewPassword>
</ChangePasswordRequest>
```

As a result, DB2 Content Manager returns an XML <ChangePasswordReply> that indicates success:

## Example: XML reply

```
<ChangePasswordReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
</ChangePasswordReply>
```

### Related information



Authenticating Web service requests for security

## Listing DB2 Content Manager entities with ListSchemaRequest

<ListSchemaRequest> helps list all entities where schemas are required.

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.

#### <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.

## Example: XML request

The following example query lists all entities on a server that require a schema, and would additionally return an attachment of the full schema:

```

<ListSchemaRequest
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>
<EntityList all="true"></EntityList>
</ListSchemaRequest>

```

As a result, DB2 Content Manager returns an XML ListSchemaReply that returns an EntityList of entities and returns the schema for each entity in the reply as an attachment.

### Example; 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>

```

#### Related information



Authenticating Web service requests for security

## Creating DB2 Content Manager items with CreateItemRequest

<CreateItemRequest> helps create items.

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.

#### <Item> (required)

Specifies an item to create on the DB2 Content Manager server.

The following example request creates a policy with one TIFF image attached, and a policy number of 57904965371.

### Example: 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_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_InsrdfName="Smith" /><XYZ_Insured XYZ_InsrdfName="Jennifer"
 XYZ_InsrdfName="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.

### Example: 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 A1001001A04I10B33015B9925018
 A04I10B33015B992501 14 1029&server=icmnlsdb&dsType=ICM"/>
</CreateItemReply>

```

The following example request creates a claim (claim number 8-123456) with one TIFF and one JPEG image attached. For this request to succeed, there must be two attachments, where the content-id for the attachments are `claimForm` and `claimPhoto`, which corresponds to the value of the name attributes of the `label` elements in the request below. If the attachments on the request have any other value for their content-id, the request fails as the corresponding attachments are not be found.

### Example: 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_ClaimForm XYZ_ClaimFName="Joannifer"
 XYZ_ClaimLName="Smith" XYZ_ClaimNumber="8-123456" XYZ_DriversLic=
 "B12004960" XYZ_InsrdfName="Nicholas" XYZ_InsrdfName=
 "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.

### Example: XML reply

```
<CreateItemReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDur1?pid=93
3 ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B33016C1746518
A04I10B33016C174651 14 1027&server=icmnlsdb&dsType=ICM"/>
</CreateItemReply>
```

#### Related information



Authenticating Web service requests for security



Working with XML services (Java only)

## Searching DB2 Content Manager items with RunQueryRequest

`<RunQueryRequest>` searches for specific items.

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.

#### `<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>`.

### `<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 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 or as an 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:

**Example: 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>icmnlsdb</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.

## Example: 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 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B33609D5857118
A04I10B33609D585711 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
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 icmnlsdb11 XYZ_Insured59 26
A1001001A04I10B33609D5857118 A04I10B33609E627501 14 1030"/>
<XYZ_Insured XYZ_InsrdFName="Jennifer" XYZ_InsrdLName="Smith"
cm:PID="91 3 ICM8 icmnlsdb11 XYZ_Insured59 26
A1001001A04I10B33609D5857118 A04I10B33609E637451 14 1030"/>
<XYZ_VIN XYZ_VIN="ICLA44P5KL9876543" cm:PID="86 3 ICM8 icmnlsdb7
XYZ_VIN59 26 A1001001A04I10B33609D5857118 A04I10B33609E646691 14
1031"/></XYZ_InsPolicy>
</ItemXML></Item></ResultSet>
</RunQueryReply>
```

You can then program your custom application to display these Web links.

### Related information



Authenticating Web service requests for security

## Retrieving DB2 Content Manager items with RetrieveItemRequest

<RetrieveItemRequest> searched and retrieves items as Web links or attachments.

To search for specific DB2 Content Manager items and retrieve them as either Web links or binary attachments, create a <RetrieveItemRequest> element 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" />
<Item URI="string" />
</RetrieveItemRequest>
```

### <RetrieveItemRequest> elements

#### <AuthenticationData> (required)

Identifies the content server (<ServerDef>), a valid user ID (<UserID>), and password (<Password>)—or a WebSphere SSO credential.

#### <Item> (required)

Specifies the item to retrieve on the DB2 Content Manager server. Each



item element specifies an item to retrieve on the DB2 Content Manager server. The item element must contain the following attribute:

**URI (required)**

Specifies the unique identifier of the item to retrieve. This is the same as the PID of the item, as stored on the DB2 Content Manager.

**<RetrieveItemRequest> attributes**

**retrieveOption (required)**

Limits the information in your search results to improve the response time. Can have one of the following values:

**IDONLY**

Checks the existence of the items in the request but not actually retrieve the data.

**ATTRONLY**

Return the items filled with only the first level of attribute 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 or as an 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).

**checkout (optional)**

Specifies whether to check out the item after it's retrieved. The default value is false.

If the item is retrieved successfully but the checkout of the item fails, the `errorOnRepeatedCheckoutDuringRetrieve` property in the `IBMCMROOT\cmgmt\cmbxmllservices.properties` configuration file determines whether the failure of the checkout operation affects the request status of the `RetrieveItemReply` message.



If `errorOnRepeatedCheckoutDuringRetrieve` property is set to true, the `RetrieveItem` response message is flagged as failed. If the `errorOnRepeatedCheckoutDuringRetrieve` property is set to false, the error during the checkout operations is ignored. The `RetrieveItem` response message is still flagged as success.

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):

### Example: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<RetrieveItemRequest contentOption="URL" retrieveOption="CONTENT"
 xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
 <AuthenticationData configString=""
 connectString="SCHEMA=ICMADMIN">
 <ServerDef>
 <ServerType>ICM</ServerType>
 <ServerName>icmnlsdb</ServerName>
 </ServerDef>
 <LoginData>
 <UserID>icmadmin</UserID>
 <Password>passw0rd</Password>
 </LoginData>
 </AuthenticationData>
 <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=91
 3 ICM6 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A07C06B71945J4200518
 A07C06B71945J420051 14 1030<server=icmnlsdb<dsType=ICM" />
 <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=92
 3 ICM6 icmnlsdb13 XYZ_InsPolicy60 26 A1001001A07C01B20133C2043418
 A07C01B20133C204342 124 1030<server=icmnlsdb<dsType=ICM" />
</RetrieveItemRequest>
```

As a result, DB2 Content Manager returns an XML `<RetrieveItemReply>` element that contains the 57904965371 policy and a URL for its content part:

### Example: XML reply

```
<?xml version="1.0" encoding="UTF-8"?>
<RetrieveItemReply
 xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
 <RequestStatus success="true"></RequestStatus>
 <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=91
 3 ICM6 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A07C06B71945J4200518
 A07C06B71945J420051 14 1030&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="91 3 ICM6 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A07C06B71945J4200518
 A07C06B71945J420051 14 1030"
 xmlns="http://www.ibm.com/xmlns/db2/cm/beans/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">
 <cm:properties type="document">
 <cm:lastChangeUserid value="ICMADMIN" />
 <cm:lastChangeTime value="2007-03-07T01:19:58.593" />
 <cm:createUserid value="ICMADMIN" />
 <cm:createTime value="2007-03-07T01:19:58.593" />
 <cm:semanticType value="1" />
 <cm:lastOperation name="RETRIEVE" value="SUCCESS" />
 </cm:properties>
 <XYZ_Insured XYZ_InsrdFName="Edward"
 XYZ_InsrdLName="Smith"
```

```

cm:PID="89 3 ICM6 icmnlsdb11 XYZ_Insured59
 26 A1001001A07C06B71945J4200518 A07C06B71959C414651 14 1031" />
<XYZ_Insured XYZ_InsrdFName="Jennifer"
 XYZ_InsrdLName="Smith"
 cm:PID="89 3 ICM6 icmnlsdb11 XYZ_Insured59
 26 A1001001A07C06B71945J4200518 A07C06B71959C707841 14 1031" />
<XYZ_VIN XYZ_VIN="ICLA44P5KL9876543"
 cm:PID="84 3 ICM6 icmnlsdb7 XYZ_VIN59
 26 A1001001A07C06B71945J4200518 A07C06B71959D546121 14 1032" />
<ICMBASE
 cm:PID="83 3 ICM6 icmnlsdb7 ICMBASE58
 26 A1001001A07C06B71946A8286318 A07C06B71946A828631 13 300"
 cm:partNumber="1">
 <cm:properties type="item"
 xsi:type="cm:partPropertyType">
 <cm:lastChangeUserid value="ICMADMIN" />
 <cm:lastChangeTime value="2007-03-07T01:19:58.593" />
 <cm:createUserid value="ICMADMIN" />
 <cm:createTime value="2007-03-07T01:19:58.593" />
 <cm:semanticType value="128" />
 <cm:lastOperation name="RETRIEVE" value="SUCCESS" />
 </cm:properties>
 <cm:resourceObject MIMEType="text/plain"
 RMName="MRMDB" SMSCollName="CBR.CLLCT001"
 externalObjectName=" " resourceFlag="2"
 size="840">
 <cm:URL value=
 "http://hostname/icmr/ICMResourceManager?order=retrieve&*amp;
 item-id=A1001001A07C06B71946A82863&*amp;version=1&*amp;
 collection=CBR.CLLCT001&*amp;libname=icmnlsdb&*amp;
 update-date=2007-03-07+01%3A19%3A58.284794&*amp;
 token=A4E6.FT6Fo_6_k_LVHc2VHR2;&*amp;content-length=0" />
 </cm:resourceObject>
 </ICMBASE>
</XYZ_InsPolicy>
</ItemXML>
</Item>
<Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=92
 3 ICM6 icmnlsdb13 XYZ_InsPolicy60 26 A1001001A07C01B20133C2043418
 A07C01B20133C204342 124 1030&*amp;server=icmnlsdb&*amp;dsType=ICM">
<ItemXML>
 <XYZ_InsPolicy XYZ_City="Temple City"
 XYZ_PolicyNum="57904965371" XYZ_State="CA"
 XYZ_Street="11132 Industry Blvd"
 XYZ_ZIPCode="90543"
 cm:PID="92 3 ICM6 icmnlsdb13 XYZ_InsPolicy60
 26 A1001001A07C01B20133C2043418 A07C01B20133C204342 124 1030"
 xmlns="http://www.ibm.com/xmlns/db2/cm/beans/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">
 <cm:properties type="document">
 <cm:lastChangeUserid value="ICMADMIN" />
 <cm:lastChangeTime value="2007-03-06T18:49:42.752" />
 <cm:createUserid value="ICMADMIN" />
 <cm:createTime value="2007-03-01T20:01:33.425" />
 <cm:semanticType value="1" />
 <cm:lastOperation name="RETRIEVE" value="SUCCESS" />
 </cm:properties>
 <XYZ_Insured XYZ_InsrdFName="Edward"
 XYZ_InsrdLName="Smith"
 cm:PID="90 3 ICM6 icmnlsdb11 XYZ_Insured60
 26 A1001001A07C01B20133C2043418 A07C01B20133E295422 124 1031" />
 <XYZ_Insured XYZ_InsrdFName="Alice"
 XYZ_InsrdLName="Smith"
 cm:PID="90 3 ICM6 icmnlsdb11 XYZ_Insured60
 26 A1001001A07C01B20133C2043418 A07C06B04942H524692 124 1031" />
 <XYZ_VIN XYZ_VIN="1RI035P5RU5435209"

```

```

cm:PID="85 3 ICM6 icmnlsdb7 XYZ_VIN60
 26 A1001001A07C01B20133C2043418 A07C01B20133E218832 124 1032" />
<ICMBASE
 cm:PID="83 3 ICM6 icmnlsdb7 ICMBASE58
 26 A1001001A07C01B20133C2122218 A07C01B20133C212221 13 300"
 cm:partNumber="1">
 <cm:properties type="item"
 xsi:type="cm:partPropertyType">
 <cm:lastChangeUserid value="ICMADMIN" />
 <cm:lastChangeTime value="2007-03-01T20:01:33.425" />
 <cm:createUserid value="ICMADMIN" />
 <cm:createTime value="2007-03-01T20:01:33.425" />
 <cm:semanticType value="128" />
 <cm:lastOperation name="RETRIEVE" value="SUCCESS" />
 </cm:properties>
 <cm:resourceObject MIMETYPE="text/plain"
 RMName="MRMDB" SMSCollName="CBR.CLLCT001"
 externalObjectName=" " resourceFlag="2" size="776">
 <cm:URL value=
 "http://hostname/icmr/ICMResourceManager?order=retrieve&
 item-id=A1001001A07C01B20133C21222&version=1&
 collection=CBR.CLLCT001&libname=icmnlsdb&
 update-date=2007-03-01+20%3A01%3A33.650668&
 token=A4E6.FT6Fo_6_WbFRyL48RAw;&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):

### Example: 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> element that contains the 8-123456 claim and binary attachments for its content parts:

### Example: 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>
<ICMBASE cm:PID="85 3 ICM8 icmnlsdb7 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>

```

#### Related information



Authenticating Web service requests for security

## Viewing your user privileges with XML requests

`<GetPrivilegesRequest>` helps view privileges on a specific item.

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.

#### `<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`.

### Example: XML request

The following example request lists all privileges for the 57904965371 policy by specifying its URI (as returned from the query search ):

```

<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> element 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.

### Example: 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>

```

#### Related information



Authenticating Web service requests for security

## 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>
 <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 URI =the PID of the item to add />
 <Folder URI = the PID of the folder to add the item to/Folder>
</AddItemToFolderRequest>

```

The <AddItemToFolderRequest> can optionally use the newVersion, checkout, and checkin attributes.

### Removing an item from a folder

```
<RemoveItemFromFolderRequest [newVersion="FALSE"
checkin="TRUE" checkout="TRUE"]>
 <AuthenticationData> ... </AuthenticationData>
 <Item URI = the PID of the item to remove/Item>
 <Folder URI = the PID of the folder to remove the item from/Folder>
</RemoveItemFromFolderRequest>
```

The <RemoveItemFromFolderRequest> can optionally use the newVersion, checkout, and checkin attributes.

### Retrieving all items in a folder

```
<RetrieveFolderItemsRequest [retrieveOption="string"
contentOption="URL" version="LATEST_VERSION"] >
 <AuthenticationData> ... </AuthenticationData>
 <Folder URI = the PID of the folder to retrieve the items from/Folder>
</RetrieveFolderItemsRequest>
```

The <RetrieveFolderItemsRequest> can optionally use the retrieveOption and contentOption 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 containing an item

```
<RetrieveFoldersForItemRequest
[retrieveOption="string" contentOption="URL"] >
 <AuthenticationData> ... </AuthenticationData>
 <Item URI = the PID of the item to retrieve the folders for/Item>
</RetrieveFoldersForItemRequest>
```

The <RetrieveFoldersForItemsRequest> can optionally use the retrieveOption and contentOption 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 categorizing the new item as a folder by setting the type attribute in the cm:properties element to folder.

### Example: 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_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:

### Example: 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:

### Example: XML request

```

<AddItemToFolderRequest
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>
<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:

### Example: 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:

### Example: 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:

### Example: XML reply

```
<DeleteItemReply
 xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
</DeleteItemReply>
```

#### Related information



Updating DB2 Content Manager items with an XML `UpdateItemRequest`



Retrieving DB2 Content Manager items with `RetrieveItemRequest`

## Updating DB2 Content Manager items with an XML `UpdateItemRequest`

`<UpdateItemRequest>` updates items.

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="PID or URL"/>
 <Add>
 <ChildItem parentURI="PID or URL">...</ChildItem>
 <Content [index="integer"]>...</Content>
 <Notelog [index="integer"]>...</Notelog>
 <Annotation [index="integer"]>...</Annotation>
 <Part attrName="String">...</Part>
 </Add>
 <Replace>
 <ItemXML>...</ItemXML>
 <Attribute name="String"> </Attribute>
 <ChildItem parentURI="PID or URL">...</ChildItem>
 <Content [index="integer"]>...</Content>
 <Notelog [index="integer"]>...</Notelog>
 <Annotation [index="integer"]>...</Annotation>
 <Part attrName="String">...</Part>
 </Replace>
 <Delete>
 <ChildItem parentURI="PID or URL">...</ChildItem>
 <Content [index="integer"]>...</Content>
 <Notelog [index="integer"]>...</Notelog>
 <Annotation [index="integer"]>...</Annotation>
 <Part attrName="String">...</Part>
 </Delete>
</UpdateItemRequest>
```

### `<UpdateItemRequest>` elements

#### `<AuthenticationData>` (required)

Identifies the content server (ServerDef), a valid user ID (UserID), and password (Password)--or a WebSphere SSO credential.



**<Item URI="PID or URL"> (required)**

Specifies the item to update in the DB2 Content Manager server.

**<Add> (optional)**

Specifies all the content to add to the item in this update request. You can add child items, content parts, notelogs, annotations, or part values for attributes.

**<Replace> (optional)**

Specifies all the content to replace in the item in this update request. You can replace the full item tree, which describes the attributes and child components of the item, by using the ItemXML element. You can also individually replace child items, content parts, notelogs, annotations, or part values for attributes.

**<Delete> (optional)**

Specifies all the content to delete from the item in this update request. You can delete child items, content parts, notelogs, annotations, or part values for attributes.

**<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.

The following example batches all three updates together to update the 57904965371 policy:

**Examples: XML request**

```
<UpdateItemRequest
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="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="1R1035P5RU5435209" 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:

### Example: 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>

```

#### Related information



Authenticating Web service requests for security

## 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>`. 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):

**Example: 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:

**Example: 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>
```

## Related information



Working with XML services (Java only)

## 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:

**Example: 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:

**Example: 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):

**Example: 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>icmnsdb</ServerName></ServerDef><LoginData>
 <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="93 3 ICM8 icmnsdb13 XYZ_InsPolicy59 26
A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029" />
<Delete>
 <ChildItem URI="91 3 ICM8 icmnsdb11 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:

**Example: 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 icmnsdb5 DOC2659 26 A1001001A04I02B22524G4418418
A04I02B22524G441841 14 1019&server=icmnsdb&dsType=ICM"/>
</UpdateItemReply>
```

## Deleting DB2 Content Manager items with DeleteItemRequest

<DeleteItemRequest> deletes items.

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.

### <Item URI=*string* /> (required)

Specifies the item to delete from the DB2 Content Manager server.

## Example: XML request

The following example deletes the 57904965371 policy by specifying its item URI (as returned from the query search):

```
<DeleteItemRequest xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
 <AuthenticationData connectionString="SCHEMA=ICMADMIN" configString="">
 <ServerDef><ServerType>ICM</ServerType>
 <ServerName>icm1sdb</ServerName></ServerDef><LoginData>
 <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
 </AuthenticationData>
 <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
 3 ICM8 icm1sdb13 XYZ_InsPolicy59 26 A1001001A04I10B44211H0825818
 A04I10B44211H082581 14 1029&server=icm1sdb&dsType=ICM" />
</DeleteItemRequest>
```

As a result, DB2 Content Manager returns an XML <DeleteItemReply> element that indicates success:

## Example: XML reply

```
<DeleteItemReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
 <RequestStatus success="true"></RequestStatus>
</DeleteItemReply>
```

### Related information



Authenticating Web service requests for security

## 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.

### <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:

### Example: 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>ecllent</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.

### Example: 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.

#### <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:

### Example: XML request

```
<CheckinRequest 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>ecllent</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.



### Example: XML reply

```
<CheckinItemReply xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
 <RequestStatus success="true"></RequestStatus>
</CheckinItemReply>
```

#### Related information



Authenticating Web service requests for security

## Linking DB2 Content Manager items with CreateLinkRequest or DeleteLinkRequest

<CreateLinkRequest> or a <DeleteLinkRequest> creates or deletes items.

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.

#### <Item> (required)

Specifies an item to create on the DB2 Content Manager server.

#### <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):

### Example: 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 A1001001A04110B44211H0825818
 A04110B44211H082581 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 A1001001A04110B44213F1658218
```

```

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.

### Example: 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):

### Example: 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.

### Example: XML reply

```

<DeleteLinkReply
xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
<RequestStatus success="true"></RequestStatus>
</DeleteLinkReply>

```

#### Related concepts

“Working with XML services (Java only)” on page 321

#### Related information

 [Authenticating Web service requests for security](#)

 [Working with XML services \(Java only\)](#)

## Moving DB2 Content Manager items between entities with MoveItemRequest

<MoveItemRequest> moves an item.

To move a DB2 Content Manager item from one entity to another (for modifying the item tree 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.

### **<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:

## **Example: 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):

### Example: XML reply

```
<MoveItemReply
 xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
 <RequestStatus success="true"></RequestStatus>
 <Item URI="91 3 ICM8 icmn1sdb11 CLAIM2_104759 26
 A1001001A04I113B04803F2085118 A04I116B41357H241001 14 1016"/>
</MoveItemReply>
```

#### Related information



Authenticating Web service requests for security

## 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, you must use the provided graphical process builder.

**Attention:** Creating document routing process with the Version 8.4 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> element 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.

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> element 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> element 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 worklist.

### <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 worklist.

### <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 355.

#### **<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.

#### **<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>yyy-mo-dd-hh.mi.ss</NotifyTime>]
 [<ResumeList><ResumeListDefinition>
 <RequiredItemTypeName>string</RequiredItemTypeName>
 <QuantityNeeded>0</QuantityNeeded>
 </ResumeListDefinition>]
 </ResumeList>]
 [<ResumeTime>yyy-mo-dd-hh.mi.ss</ResumeTime>]
 [<TimeLastMoved>yyy-mo-dd-hh.mi.ss</TimeLastMoved>]
 [<UserLastMoved>string</UserLastMoved>]
 [<ProcessCompletionTime>yyy-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: *yyy-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 value of <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.

**Related information**

Authenticating Web service requests for security

**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>
```

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>
```

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>
```

## **<ListActionRequest> elements in an ActionList**

### **<AuthenticationData> (required)**

Identifies the content server (<ServerDef>), a valid user ID (<UserID>), and password (<Password>)--or a WebSphere SSO credential.

### **<ActionListName> (required)**

Specifies the name of the action list to list the action names for.

## **<ListActionRequest> elements to list the values of an ActionName**

### **<AuthenticationData> (required)**

Identifies the content server (<ServerDef>), a valid user ID (<UserID>), and password (<Password>)--or a WebSphere SSO credential.

### **<ActionName> (required)**

Specifies the name of the action to list.

## **<Action> elements**

### **<ActionName> (required)**

Specifies the name of the action.

### **<Predefined> (required)**

If set to true, then this action is a defined action in DB2 Content Manager.  
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.

## **Related information**



Authenticating Web service requests for security

## **Updating a work package with UpdateWorkPackageRequest**

<UpdateWorkPackageRequest> updates objects in a work package.

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.

### **Example: 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**

`<StartProcessRequest>` starts a document routing process.

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.

#### **<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.

### **Example: 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.

#### Related information

 Authenticating Web service requests for security

### 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.

##### <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.

#### Example: 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.

#### Related information

 Authenticating Web service requests for security

### Continuing a process with ContinueProcessRequest

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.

#### **<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.

### **Example: 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>
```

The following example query assigns user-defined `<ContainerData>` to the specified work package, and continues it along the process.

As a result, DB2 Content Manager returns an XML `<ContinueProcessReply>` that indicates success.

### **Related information**



Authenticating Web service requests for security

## 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.

#### <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.

### Example: XML request

```
<SuspendProcessRequest 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 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.

### Related information



Authenticating Web service requests for security

## Resuming a process with `ResumeProcessRequest`

You can force a suspended process to resume by specifying the associated work package in a `<ResumeProcessRequest>` element. 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>` element), a valid user ID (`<UserID>`), and password (`<Password>`)--or a WebSphere SSO credential.

#### `<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.

### Example: 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.

### Related information

## 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.

#### `<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>`:

### Example: 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 A1001001A04110B44211H0825818
 A04110B44211H082581 14 1029&server=icmnlsdb&dsType=ICM" />
 </DeleteItemRequest>

 <DeleteItemRequest
 xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
```

```

 <AuthenticationData connectionString="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.

### Example: 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>

```

#### Related information



Authenticating Web service requests for security

---

## Working with the Web services

You can use a Web service within your applications, with other Web services interfaces, or in complex business processes to perform actions with a DB2 Content Manager system.

---

### Web services overview

Web services 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 markup 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 content, such as integers, that are often represented differently by the various hardware and software programming languages that are used today. XML is language and operating system independent, which saves time and resources when you are integrating applications with DB2 Content Manager

Simple Access Object Protocol (SOAP) is an XML-based messaging protocol that is used for Web services 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 (such as 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 functions such as security, priority, and auditing.

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 services interface. WSDL describes the location of the Web service, how to connect to it, which parameters must be passed in the SOAP request, which values to return, binding information, and so on.

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 change with the inevitable software changes that are 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**

Reduces staffing requirements, replaces paper processing, and reduces errors.

**Leverage existing investments**

You can use old software in new ways by building a Web services layer for universal access.

To work with the DB2 Content Manager Web services, you must have a working knowledge of the Web services technology. To find out more about the Web services standards, see World Wide Web Consortium (W3C).

## DB2 Content Manager Web services

A *Web service* interface is a reusable, loosely coupled, software component that can be located, published, and invoked through a network, like the Web.

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.

With the Web services interface, you can dynamically integrate your applications with DB2 Content Manager, regardless of the programming language that they were written in and the operating system that they run on. 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 interface 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 customers to view the current Blue Book value of their car.

### The Web services features in DB2 Content Manager Version 8.4

The important features of DB2 Content Manager Version 8.4 Web services are:

- Supports only WebSphere Application Server Version 6.1. It no longer supports WebSphere Application Server version 5.1.1 and 6.0.x anymore.
- Supports WebSphere Network Deployment and WebSphere server cluster.
- Uses any application server that an installation user specifies. The Web services configuration program does not create a designated application server



(cmwebsvc) before deploying Web services application. See WebSphere documentation to create the server if the server doesn't already exist.

- Supports updated Websphere connection pool.

The installation users must configure the Web server for Web services application. The Web services configuration program does not configure the Web server in 8.4.

---

## 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. The document that is passed contains all of the details needed to specify the operation to be performed and the objects on which that the operation is to be performed.

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 self-describing, self-validating, and readable by people. 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 does not fail.

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 needed if the exchange conforms to the XML document format. One reason for not using object serialization is to support client and server 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 persists on the server.

## DB2 Content Manager Web services architecture

The main components in the DB2 Content Manager Web services architecture include the requester, the Web services server, the XML beans layer, and the DB2 Content Manager repository.

Here is the general interaction flow of the main components of the Web services:

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 and extracts the binary data being uploaded from SOAP attachments.
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 exception information (if applicable). If the APIs return any binary, they are not contained in the message but referred separately.



7. The message is returned to the DB2 Content Manager Web services server.
8. The Web services server creates a SOAP message.
9. The SOAP message is returned to the requester.

Figure 31 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 service mid-tier, which includes the web services server, web services servlet, XML Beans, and APIs processes the SOAP request and sends the data to the library server. The claim is stored into the library server and pictures associated with the claim are stored in the resource manager.

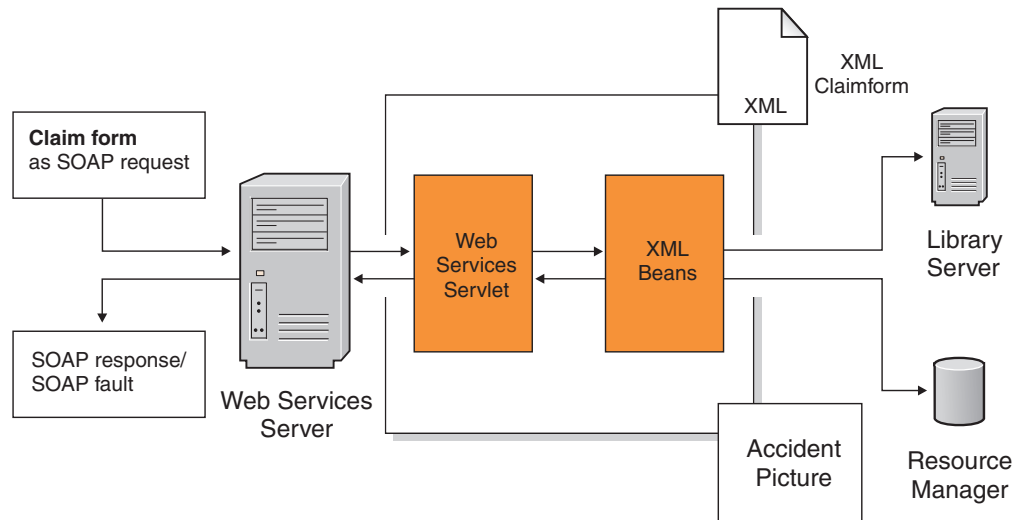


Figure 31. Processing a document using Web services

## Comparison of the DB2 Content Manager Web services interfaces

DB2 Content Manager offers the same Web services functionality through two different interfaces: CMBGenericWebService and CMWebService according to the different requirements for integrating DB2 Content Manager Web services into customer applications.

The interfaces of CMWebService defines all the operations in detail in the WSDL. The WSDL also refers to XML schemas that define the request and response messages for each operation and the constructing elements for these messages. When the WSDL and XML schemas are translated into proxy classes of a particular programming environment, the proxy classes can provide APIs to help build request messages and decompose the response messages for client applications. Therefore, the client developers do not need to assemble or disassemble the XML messages manually. The proxy class APIs are also strong-typed, which prevents the errors that can happen during assembling or disassembling the XML message manually. This is called the generator style of a Web services interface.

Optionally, you can use XML schema for specified DB2 Content Manager item types to generate client proxy classes. In such a case, the proxy class APIs are detailed to such an extent that the client application developers do not need to know anything about the syntax of the XML messages. The developers can follow the object-oriented programming guidelines to use these proxy classes to assemble request XML messages and disassemble response XML messages that leads to a

rapid development in the client applications. But it also means that if the item types are changed, added, or removed from the definition, the developers must get the new XML schema for the item types, regenerate the proxy classes, and recompile the application with the new classes.

However, if the XML schema for the specific item types is not included when generating the proxy classes with CMWebServices WSDL, the client application developers must write their own code to serialize the XML elements for the item types in the request messages and de-serialize the XML elements for the item types in the response messages. The benefit of not including the specific item types in the proxy classes is that the client applications have the flexibility to work with any item types.

The WSDL of the CMBGenericWebServices defines the generic style of Web services interfaces. It does not define the Web services operations and the request and response message. Instead, it provides a generic API so that any form of request XML messages can be sent to the DB2 Content Manager Web services server, and it can receive any form of response XML messages from the server. This generic, or late-binding, style of Web services interfaces gives the freedom to the DB2 Content Manager Web services server to add, change, or remove the Web services operations and their parameters in the future, but the client application developers are not forced to get the new WSDL and recompile their application.

However, because of this flexibility of the CMBGenericWebServices interface, the client application developers must have full knowledge of the Web services operations and the request and response XML messages. They must assemble or disassemble the message manually. They can use references, such as the WSDL to CMBGenericWebServices, the XML schema for messages defined in `cmbmessages.xsd`, the XML schema for elements defined in `cmdatamodel.xsd` for constructing item type XMLs (both XML schema definition files are located in `IBMCMROOT/config` where `IBMCMROOT` is the directory where DB2 Content Manager Information Integrator for Content is installed) and the Web services samples that come with DB2 Content Manager Information Integrator for Content.

In terms of SOAP message style, both CMWebService and CMBGenericWebService use document/literal, which means the request and the response are document-style messages and are not encoded.

For a particular application, you should consider the SOAP attachment capacity when choosing the Web services endpoint.

The attachment types supported by CMWebServices are:

- Direct Internet Message Encapsulation (DIME)
- Multipurpose Internet Mail Extensions (MIME)
- Message Transmission Optimization Mechanism (MTOM)

At run time, only one of the three SOAP attachment formats is enabled by configuration. The attachment type supported by CMBGenericWebServices is MIME.

Many of the tools for Web services development generate client code, sometimes called proxy classes, to represent the Web service interface and operations supported by the Web service. This allows the details of building SOAP messages, attachments, socket programming, and so forth to be hidden so that the client has to build and send requests only to a local proxy version of the Web service.

DB2 Content Manager Web services server responds to any request messages that comply to the interfaces, and the response messages that are coming out from it are based on the interfaces too. The client application developers decide how to generate and understand the messages, including choosing which Web services endpoints to visit and what development environment and programming language to use. But the different client development tools have their own strength and limitations. The choice is based on attachment type:

- Microsoft Visual Studio .NET 2003, which uses .NET Framework SDK version 1.1 and Web Services Enhancements (WSE) SDK version 1.0 sp1 supports only DIME attachments
- Microsoft Visual Studio 2005 using Microsoft .NET Framework SDK version 2.0 or 3.0 and Web Services Enhancements (WSE) SDK version 3.0 supports only MTOM
- Rational® Application Developer version 6.0 and WebSphere client tools, support only MIME

Among non-IBM Web services software, Apache Axis supports both DIME and MIME. Apache Axis2 supports both MIME and MTOM. Therefore, depending on your client development environment, you must choose the matching Web service endpoint (CMWebService or CMBGenericWebService) to connect to based on the SOAP message capacity.

To see the code differences between the two interfaces, see the sample programs included with IBM DB2 Information Integrator for Content.

#### **Related concepts**

“Web services samples” on page 437

A set of thoroughly commented Web services samples are provided to help you work with the Web services.

#### **Related tasks**

“Generating the WSDL file for the CMWebService Web service from system administration client” on page 420

Exporting item types to WSDL is still supported in DB2 Content Manager Version 8.4 system administration client. It generates the WSDL and the supporting schema files for a group of item types.

## **Web services functions**

The Web services do not support the full set of DB2 Content Manager operations.

The functions supported by the Web services include the most commonly used runtime functions, which do not include administration and data modeling. The following list contains the operations supported by the Web services:

- Create, Retrieve, Update, and Delete items
- Query
- Folder operations
- Link operations
- Check in and Check out items
- Change user password
- List user privileges for an item
- List available servers (library servers)
- List item types and schemas: Retrieve an XML schema definition for one or all item types.

- Move an item: Change an item's metadata.
- Document-routing operations
- Batch: Group several operations into a single request (one round trip to the server)
- Transactions: Group several requests within a batch request into a single transaction with changes rolled back if an error occurs

#### **Related reference**

"List of Web services requests"

In the Web services, operations that allow you to connect, create, query, retrieve, and modify data in the system take the form of requests.

## **List of Web services requests**

In the Web services, operations that allow you to connect, create, query, retrieve, and modify data in the system take the form of requests.

You can create Web services requests according to the schema defined in the `cmbmessages.xsd` and `cmdatamodel.xsd` files. You can find the request in the schema to search for the request name. The schema for this request describes the parameters that the operation requires. You can find the file in the `IBMCMROOT\config` directory.

You can make the following requests through the Web services:

#### **AddItemToFolderRequest**

Adds an existing item into an existing folder.

#### **BatchRequest**

Allows you to group independent requests into a single request, performed in one trip to the server. The requests are performed sequentially on the server, and the individual replies for each request are grouped into a batch reply.

The batch request can contain many different request types, many instances of the same request type, or any combination. You can create a batch request that is 10 `RetrieveItemRequests` for 10 different items (in effect, a bulk retrieve) or you can create a batch request with a `RetrieveItemRequest`, followed by an `UpdateItemRequest`, followed by a query, then a `ListWorkNodesRequest`, then five more `RetrieveItemRequests`, and so forth.

#### **ChangePasswordRequest**

Enables you to change a user's password.

#### **CheckInItemRequest**

Enables you to check in an item that is currently checked out.

#### **CheckOutItemRequest**

Enables you to check out an item from the server.

#### **CreateItemRequest**

Allows you to create a new item on the server. You must pass in an XML document conforming to the schema for the specified item type, and the data that will become the content of the new item must be passed in as attachments to the request.

#### **CreateLinkRequest**

Creates a link between two existing items.

**DeleteLinkRequest**

Deletes a link between two items.

**DeleteItemRequest**

Deletes an item from the server.

**GetPrivilegesRequest**

Returns a list of the access privileges the current user has for a particular item.

**ListSchemaRequest**

Returns a list of the item types (entities) that are available to a user and includes the XML schema for that item type as an attachment on the reply. You can specify one or more specific item types whose schemas should be returned, or you can get the schemas for all item types that the user specified in the request has access to.

**ListServersRequest**

Returns a list of all servers of a particular type, such as ICM, that are available for users to connect to. This is the only request that does not require any user authentication because it is not necessary to connect to retrieve this information.

**MoveItemRequest**

Moves, or re-indexes, an item from one item type to another. The metadata for the new item must be given as an XML document fragment conforming to the schema of the new item type.

**RemoveItemFromFolderRequest**

Removes an existing item from a folder on the server.

**RetrieveFolderItemsRequest**

Returns a list of items contained within a folder. You can choose to return just the document IDs of the items, their metadata, or all of their data, including content.

**RetrieveFoldersForItemRequest**

Retrieves the list of document IDs of all folders that contain the given item.

**RetrieveItemRequest**

Retrieves the content of an item. You can choose to retrieve either just the metadata for the item, to retrieve the content for the item and its metadata, or to retrieve the links, the content, and metadata.

**RunQueryRequest**

Runs a query string given in the request and returns the results of the query. You can choose to get only a list of document IDs in the reply, or you can choose to retrieve any level of content, from metadata, metadata and content, or metadata, content, and links.

**UpdateItemRequest**

Updates an item. You can choose to add and remove child components, notelogs, annotations, or parts in a document. You can also replace the item's metadata values by specifying new values for the updated item in an XML document fragment conforming to the schema for the item type, and replace existing child components, notelogs, annotations, and parts. The document ID for the item is returned in the reply because in some cases updating the item changes the document ID.

## Document routing requests

The following list describes the document routing requests:

### **ContinueProcessRequest**

Moves the specified work package to the next work node. The work package will be removed from the server and a new work package or many work packages (in the case of a parallel routing) will be created for the specified work package owner.

### **ListActionRequest**

Lists the action for a specified action name.

### **ListActionNamesRequest**

Lists the action names for a specified action list.

### **ListNextWorkPackageRequest**

Retrieves the next work package in the specified work list for the specified owner and checks out the referenced item. The reply gives the next work package in which the referenced item in the work package is not checked out by another user.

### **ListProcessRequest**

Lists the process for the given process name.

### **ListProcessesRequest**

Lists all the processes.

### **ListProcessNamesRequest**

Lists process names.

### **ListWorkListNamesRequest**

Lists the worklist names.

### **ListWorkListsRequest**

Lists the worklists.

### **ListWorkNodeRequest**

Lists the work node specified by the work node name.

### **ListWorkNodesRequest**

Lists the work nodes.

### **ListWorkPackageCheckOutOptionRequest**

Lists the work package attributes for the given work package URI. The item associated with the work package can be checked out depending on the value specified for the element checkOutItem.

### **ListWorkPackagesRequest**

List all workpackages for the specified worklist name and work package owner.

### **ResumeProcessRequest**

Resumes a process. When a process is resumed, the work package attribute suspend flag is set to false. There is no routing of the work package and the item associated with the work package is not checked out.

### **SuspendProcessRequest**

Suspends a process with the specified work package for the number of minutes specified in the request for the given resume list.

### **StartProcessRequest**

Start an item on a document routing process.

### **TerminateProcessRequest**

Terminates a process. The work package specified will be removed from the server. The item referenced in the work package will be checked in if it is checked out.

### **UpdateWorkPackageRequest**

Updates a work package's container data, work package owner, and work package priority.

#### **Related concepts**

"Web services samples" on page 437

A set of thoroughly commented Web services samples are provided to help you work with the Web services.

---

## **Installing and configuring the DB2 Content Manager Web services**

The following topics guide you through the installation and configuration of DB2 Content Manager Version 8.4 Web services.

### **Platforms supported by the Web services**

You can use the Web services on various platforms.

### **Software prerequisites for the DB2 Content Manager Web services server**

WebSphere Application Server Version 6.1 Base and Network Deployment

**Important:** DB2 Content Manager V8.4 does not support WebSphere Application Server Version 5.1.1 and WebSphere Application Server Version 6.0.2.

### **Software prerequisites for a DB2 Content Manager 8.4 Web services client**

#### **Java clients**

- Java client applications to CMBGenericWebService must be developed with WebSphere Application Server 6.1.
- Java client applications to CMWebService with a DIME or MTOM attachment type must be developed with Apache Axis version 1.1 or above
- Java client applications to CMWebService with MTOM attachment type must be developed with Apache Axis2 version 1.1.1

#### **.NET 1.1 client**

- Microsoft .NET Framework SDK 1.1
- Microsoft Web Services Enhancement (WSE) SDK 1.0 service pack 1
- Microsoft Hotfix 892202

You must contact Microsoft for the hotfix. If you do not have the hotfix installed, you will receive errors when generating the WSDL.

#### **.NET 2.0 client**

- Microsoft® .NET Framework SDK 2.0 or 3.0
- Microsoft Web Services Enhancement (WSE) SDK 3.0

**Important:** Web services for DB2 Content Manager Version 8.4 does not support WebSphere Application Server Network Deployment in Linux for system Z.



### **Related concepts**

“Supported WebSphere Application Server topologies”

This section provides a brief description of the WebSphere Application Server topologies supported by the DB2 Content Manager Web services.

## **Installation and configuration of Web services**

When you install, the installation program copies the program files from installation media to local hard drive. When you configure, you create necessary configuration files and databases, and deploy Web applications on WebSphere Application Servers.

There are several significant changes in the installation of Web services from DB2 Content Manager Version 8.3 to DB2 Content Manager Version 8.4.

In DB2 Content Manager Version 8.3, the installation and configuration was done using a single user interface (GUI). For example:

1. DB2 Information Integrator for Content installation program prompts the user to select components to be installed (connectors, Web services, infocenter, and so on.).
2. The user must input configuration parameters for selected components.
3. The installation program copies the program files for the selected components to the local hard drive.
4. After the files are copied, the installation program opens the configuration programs to configure the selected components. For Web services, the configuration deploys the Web services application onto WebSphere Application Server.

DB2 Content Manager Version 8.4 divides the installation and the configuration into two separate programs. The installation program copies only the program files from installation media to the local hard drive. The configuration program performs configuration tasks for installed components. You can open each of the programs separately as often as necessary. For example, you can configure an installed component multiple times without installing and uninstalling it. You can also uninstall or reinstall a component without uninstalling or reinstalling the whole product.

The installation and configuration of Web services using two separate GUIs provides flexibility because it is one of components in DB2 Information Integrator for Content product in version 8.4.

Web services depends on ICM connector, which is a component of DB2 Information Integrator for Content. Therefore, ICM connector must be installed if you choose to install Web services when you are installing DB2 Information Integrator for Content. ICM connector must also be configured before configuring Web services.

### **Upgrading Web services from Version 8.3 to 8.4**

DB2 Content Manager Version 8.4 supports upgrading Web services Version 8.3 to Version 8.4. For more information, see [Upgrading Web services to Version 8.4](#)

## **Supported WebSphere Application Server topologies**

This section provides a brief description of the WebSphere Application Server topologies supported by the DB2 Content Manager Web services.



## Deployment target

Web services is one of the components of DB2 Information Integrator for Content. In DB2 Content Manager Version 8.3, when you are configuring the Web services (deploying Web services application to WebSphere Application Server), the application node must reside on the same machine where DB2 Information Integrator for Content is installed, which means that the deployment target (application server node) must be local to DB2 Information Integrator for Content. Therefore, if DB2 Information Integrator for Content is installed on a machine where there is only a deployment manager node, the Web services application cannot be deployed to any application server node in the WebSphere Network Deployment environment managed by this deployment manager.

However, DB2 Content Manager Version 8.4 Web services supports WebSphere Network Deployment. Therefore, Web services application can be deployed on a remote application server node, which does not need to be on the machine where DB2 Information Integrator for Content is installed if there is a node (either deployment manager node or application server node federated to the deployment manager) on the machine. The WebSphere Network Deployment management is responsible for transferring the application to the target node that can be on a remote machine. DB2 Content Manager Version 8.4 Web Services application can run depending on local resources only. It does not refer to any information on the machine where DB2 Information Integrator for Content is installed.

## Supported topologies

The following topologies are supported by DB2 Content Manager Version 8.4.

### Topology A - Unmanaged node

An unmanaged node is an island. It is a WebSphere application server node with its own administration facilities. In this topology, the DB2 Information Integrator for Content must be installed on this machine. The unmanaged node can be created either by installing a WebSphere Application Server Base package or Network Deployment package. In the latter case, the node is not federated to the deployment manager.

### Topology B - Managed node

A managed node is an application server node that is already federated to a deployment manager. This node can be local or remote to the machine where the DB2 Information Integrator for Content is installed. If it is local, it has the same topology as in unmanaged node. If it is remote, the DB2 Information Integrator for Content installed machine must also have a node (deployment manager node or application server node federated to the deployment manager). The machine where the DB2 Information Integrator for Content is installed and the machine where the managed node is must be in the same management hierarchy of Network Deployment.

In both topologies, A and B, an application server name must also be supplied for specifying a deployment target besides the node name. Because the Web services configuration program does not create a designated server, the server must be already existing at the time when the application is deployed.

### Topology C - Cluster

A server cluster is a group of application servers on different application server nodes that share the load for particular applications. This topology

is similar to the managed node scenario. The difference is that the deployment target consists of multiple application server nodes. These nodes are grouped into a cluster. The cluster name is used as the deployment target. The deployment manager automatically deploys the Web services applications onto each member servers in the cluster when the cluster name is used.

You must be able to reach the cluster from the machine where DB2 Information Integrator for Content is installed. The machine that has DB2 Information Integrator for Content installed must also have a node (deployment manager node or application server node federated to the deployment manager).

Upgrading Web services from DB2 Content Manager Version 8.3 to Version 8.4 does not support cluster. For more information, see Upgrading Web services to Version 8.4.

**Important:** Web services for DB2 Content Manager Version 8.4 does not support WebSphere Application Server Network Deployment in Linux for system Z.

## Verifying, starting, and stopping the Web services

After you install the Web services, you must verify that they were installed correctly, start, and stop them before you can work with them.

To start the Web services application, you must start the WebSphere Application Server or cluster on which the application is deployed.

The Web services application is stopped if you stop the WebSphere Application Server or cluster on which the application is deployed.

To verify if the Web services application is installed correctly, you must start the application first. After starting the application, connect to the following URLs in a Web browser.

- <http://hostname/CMBSpecificWebService/services/CMWebService>
- <http://hostname/CMBGenericWebService/services/CMBGenericWebService>

You should see generic text that is not an error message.

**Restriction:** If you do not have the HTTP Server installed, you must add the port used by the WebSphere Application Server to your host name. For example: <http://hostname:9082/CMBGenericWebService/services/CMWebService>.

 Starting an application server

 Stopping an application server

 Starting clusters

 Stopping clusters

## Post installation configuration

After the Web services is installed and deployed on WebSphere Application Server, the following tasks must be performed before it is functional.

## Configuring shared library for JDBC driver

DB2 Content Manager Version 8.4 Web services provides different solutions for different types of JDBC driver configurations.

DB2 Content Manager Web services calls ICM connector APIs. The ICM connector API depends on JDBC driver to connect to library server databases. For DB2 Content Manager Version 8.4 Web Services for multiplatforms with DB2, the JDBC driver is shipped with Information Integrator for Content. Therefore, the JDBC driver JAR files are packaged into the Web services EAR file before it is deployed on WebSphere Application Server. After the deployment, the JDBC driver JAR files are in:

`{WAS_HOME}/profiles/{profileName}/installedApps/{cellName}/{appName}/lib`, where *WAS\_HOME* is the IBM WebSphere Application Server home directory, *profileName* is the name of the directory of the profile where the Web services application is installed, *cellName* is the name of the WebSphere cell where the Web services application is installed, and *appName* is the WebSphere application name specified for the Web services application.

This directory is in the class path of the Web Services application at run time. ICM connector APIs can locate the JDBC in the class path.

For DB2 Content Manager V8.4 Web services for zOS with DB2 Universal Database, the JDBC driver JAR files are not packaged in the Web services EAR file. The Web services configuration program creates the following two shared libraries after deploying the application:

### **sharedLibrary\_1\_name=DB2 JDBC**

```
sharedLibrary_1_classpath=${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
sharedLibrary_1_nativepath=${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}
```

### **sharedLibrary\_2\_name=DB2 JDBC2**

```
sharedLibrary_2_classpath=${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
sharedLibrary_2_nativepath=${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}
```

The shared libraries use the following two existing WebSphere variables in their paths:

- `DB2UNIVERSAL_JDBC_DRIVER_PATH`
- `DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH`

The administrator of DB2 Content Manager V8.4 Web services application must ensure that the value of these two variables is set to the directories where the JDBC driver is actually installed.

## Updating configuration files

When configuring Web services, you must copy the files to the locations where Web services application is deployed.

The following files are copied to the locations where Web services application is deployed:

- `IBMCMROOT/cmgmt/cmbcmenv.properties`
- `IBMCMWorkingDirectory/cmgmt/cmbxmlservices.properties`

- IBMCMWorkingDirectory/cmgmt/cmbwebservices.properties
- IBMCMWorkingDirectory/cmgmt/cmbconnectionpool.properties
- IBMCMWorkingDirectory/cmgmt/cmbcmenv.properties
- INI configuration files for DB2 Content Manager connector that are in the directory specified by CMCFGDIR property in file IBMCMROOT/cmgmt/cmbcmenv.properties

where *IBMCMROOT* is the install directory of IBM DB2 Information Integrator for Content and *IBMCMWorkingDirectory* is the property defined in file *IBMCMROOT/cmgmt/ibmcmconfig.properties*.

Each WebSphere application server on which Web services application is deployed has a copy of these files, with the modification for the particular runtime environment in the application server.

To synchronize these copies with the content of the originals in the list above, you must use IBM DB2 Information Integrator for Content configuration wizard to configure Web services again.

**Remember:** You must enter the exactly same parameters you used last time for configuring Web services.

## Configuring WebSphere Application Server connection pooling for Web services

WebSphere Application Server connection pooling enables a datastore to scale to an indefinite number of users. Connection pooling also improves Web services performance by reducing logon time and memory usage.

### Important:

For Web services, the cmbpool.ini file is in *\${WAS\_HOME}/profiles/\${profileName}/installedApps/\${cellName}/\${appName}/cmgmt/connectors* and not in */home/ibmcmadm/cmgmt/connectors* directory for AIX, Linux, or Solaris, or in the *IBMCMROOT\cmgmt\connectors* for Windows.

## Logging and tracing

In DB2 Content Manager Version 8.4, the Web services application has its own log file on each application server node.

DB2 Content Manager Version 8.3 Web services application writes its log messages into beans log, which is *IBMCMROOT/log/beans/\${user.name}.beans.log*.

In Version 8.4, the Web services application has its own log file on each application server node. The log files are in *\${SERVER\_LOG\_ROOT}/\${appName}/websvc*, where *SERVER\_LOG\_ROOT* is a WebSphere variable that is defined in application server scope and *{appName}* is the application name chosen when deploying the Web services application. The default value of the *SERVER\_LOG\_ROOT* is *\${WAS\_HOME}/profiles/\${profileName}/logs/\${serverName}*, where *serverName* is the name of the application server on which the Web services application is deployed.

The beans log files are in *\${SERVER\_LOG\_ROOT}/\${appName}/beans..*

The OOAPI log files are in *\${SERVER\_LOG\_ROOT}/\${appName}/connectors..*

These directories are configured in `${WAS_HOME}/profiles/${profileName}/installedApps/${cellName}/${appName}/cmgmt/connectors/cmblogconfig.properties`.

Besides `cmblogconfig.properties`, `${WAS_HOME}/profiles/${profileName}/installedApps/${cellName}/${appName}/cmgmt/cmbwebservices.properties` has two properties controlling the logging and tracing at Web services layer only. The properties are:

**messageTracingEnabled**

For debugging Web services client. The default value of `messageTracingEnabled` is false.

**Important:** When this option is true, it affects Web services performance. When this option is set to true and when the trace level for the Web services component in `cmblogconfig.properties` common logging configuration file is set to `DEBUG`, the Web services layer messages are logged into Web services log file.

**prependLoggerNameToMessage**

Determines whether to add class name before log message in Web services log file. The default value of `prependLoggerNameToMessage` is true.

## Un-installing and removing configuration

You can un-install Web services by using the installation application for DB2 Information Integrator for Content. You can remove Web services configuration by using the configuration application for DB2 Information Integrator for Content.

Removing Web services configuration results in Web services enterprise application being un-installed from WebSphere Application Server Version 6.1. The files that were not modified in the directory where Web services enterprise application was installed are removed. If a file is modified since it was deployed in WebSphere Application Server, the file is not removed, for example, the configuration files and the log files.

The configuration files in the directory where Web services application is installed are copies of the files in `IBMCMROOT/cmgmt` and `IBMCMWORKINGDIRECTORY/cmgmt`, where `IBMCMROOT` is install directory for DB2 Information Integrator for Content and `IBMCMWORKINGDIRECTORY` is defined in `IBMCMROOT/cmgmt/ibmcmconfig.properties`. If you change the configuration to impact the behavior of the Web services application, you must change the files in `IBMCMROOT/cmgmt` and `IBMCMWORKINGDIRECTORY/cmgmt` and use the WebSphere administration console to refresh the configuration files in the directory where Web services application is installed.

**Important:** You should not directly change the configuration files in the directory where Web services application is installed because all the changes that you made are lost when you re-deploy Web services application.

**Important:** You must save a copy of the log files in Web services application installed directory before un-installing or re-installing Web services application to WebSphere Application Server Version 6.1 to store the debugging and tracing information from the previous installation.

## Integrating Web services into your applications or processes

You can integrate your client application or business process by using the Web services.

You can develop client applications to interact with the following DB2 Content Manager Version 8 Release 4 Web services interfaces:

- CMWebService
- CMBGenericWebService

### Integrating CMWebService

To integrate CMWebService, you must create the WSDL file and its supporting XML schema files, generate proxy classes for WSDL, and write your application to use those proxy classes.

When you use the CMWebService, the main steps that you perform are as follows:

1. Create the WSDL file and its supporting XML schema files from the system administration client or by using a command-line tool, or from the URL:  
`http://hostname/CMBSpecificWebService/services/CMWebService/CMWebService.wsdl`.  
The WSDL and the XML schema files contain all of the available operations supported by the Web service and any item type definitions that you can include.
2. Generate proxy classes for the WSDL file and its supporting XML schema files.
3. Write your application program to use the proxy classes to communicate with the CMWebService. The CMWebService URL is `http://hostname/CMBSpecificWebService/services/CMWebService`.

There are three Web services client samples in DB2 Content Manager Version 8.4 that demonstrate how to develop Web services clients to specific Web services (CMWebService) in C# and Java.

The C# sample with .NET1.1 is in *IBMCMROOT/samples/webservices/CMWebService/CS/WSE1/base*. The sample requires the Web service support and WSDL utility provided by Microsoft .NET 1.1 SDK, WSE 1.0 SP1 SDK and Visual Studio .NET 2003. For more information, see the sample readme file in *IBMCMROOT/samples/webservices/CMWebServiceClient.readme*.

The C# sample with .NET 2.0 is in *IBMCMROOT/samples/webservices/CMWebService/CS/WSE3/base*. It requires the Web service support and WSDL utility provided by Microsoft .NET 2.0 SDK, WSE 3.0 SDK and Visual Studio 2005. For more information, see the sample readme file in *IBMCMROOT/samples/webservices/CMWebServiceClientWSE30.readme*.

The Java sample in *IBMCMROOT/samples/webservices/CMWebService/java/Axis/base* demonstrates how to develop a Java client to CMWebService. It requires JDK 1.5 and Apache Axis Version 1.4. For more information, see the sample readme file in *IBMCMROOT/samples/webservices/CMWebServiceClientJava.readme*.

#### Related information



Getting started with the CMWebService in a .NET environment



## Generating the WSDL file for the CMWebService Web service

You can generate a WSDL file by using the system administration client or from a command line. However, it is more convenient to download from the URL of the endpoint, with appropriate parameters in the query string of the URL of CMWebService endpoint.

In DB2 Content Manager Version 8.4, the interfaces of CMWebService Web services are defined in the following files:

### **CMWebService.wsdl**

Main definition file for description specific Web services endpoint (CMWebService). It defines the port types (such as. the operations), the names of input and output messages for each port type, and the binding for the Web services endpoint. The WSDL imports cmbmessages.xsd for the detailed definitions of the messages.

### **cmbmessages.xsd**

Schema of the XML representation of the input and output messages for the operations of specific Web services endpoint (CMWebService). It imports cmdatamodel.xsd to define the details for the XML representation of DB2 Content Manager Version 8.4 data model in messages. Optionally, it includes the itemtype.xsd file if the item types are specified when generating the cmbmessages.xsd file. In this case, the cmbmessages.xsd file assumes that the itemtype.xsd has the XML schema declarations for these item types and the cmbmessages.xsd file refers to these declarations.

### **cmdatamodel.xsd**

Schema for the XML representation for DB2 Content Manager Version 8.4 data model.

### **itemtype.xsd**

Schema for the XML representation of the selected item types. It imports the cmdatamodel.xsd file to define the XML representation of DB2 Content Manager Version 8.4 data model.

You can download the files from the following well formed URLs of the Web services endpoint.

### **CMWebService.wsdl**

<http://hostname/CMBSpecificWebService/services/CMWebService/CMWebService.wsdl>

### **cmbmessages.xsd**

<http://hostname/CMBSpecificWebService/services/CMWebService/cmbmessages.xsd?itemtype=itemtype1,itemtype2,itemtype3&b=true>, where

### **itemtype(optional)**

String with concatenated item type names separated by a comma. If the parameter is presented in the URL query string, the downloaded cmbmessages.xsd file has an `xs:include` element whose `schemaLocation` attribute contains a relative URL pointing to the itemtype.xsd file. It assumes that the itemtype.xsd file has the XML schema declarations of the item types whose names are specified in the parameter and it refers to these item type names. The item type names must be the short names of these item types instead of their display names.

If the parameter is not in the query string, the cmbmessages.xsd file does not include the itemtype.xsd file and does not refer to any item type names.

The value of the parameter cannot be validated in anyway. Set it with correct item type names so that it can correctly refer to the XML schema declarations in the itemtype.xsd file.

**b(optional)**

Parameter for the backward compatibility for users who have the Web services client applications connected to CMWebService endpoint, which were developed with DB2 Content Manager Version 8.3 Fix Pack 4 or earlier. Without any change, these client applications can work well with DB2 Content Manager 8.4 Web services server. If you want to regenerate the proxy classes using new WSDL and schema files downloaded from DB2 Content Manager 8.4 Web services server and do not want to change the other parts in their applications that use the proxy classes, you can download the cmbmessages.xsd file by using the parameter with the value true. If the parameter is not set or its value is not true, the downloaded cmbmessages.xsd file generates incompatible proxy classes for those client applications mentioned above.

**cmdatamodel.xsd**

http://hostname/CMBSpecificWebService/services/CMWebServices/  
cmdatamodel.xsd

**itemtype.xsd**

http://hostname/CMBSpecificWebService/services/CMWebService/  
itemtype.xsd?itemtype=itemtype1,itemtype2,itemtype3&server=ICMNLSDDB  
&credential=true&b=true, where

**itemtype(required)**

See the explanation to the URL for downloading the cmbmessages.xsd file. The Web services will extract the specified item types and save their XML schema declarations in the itemtype.xsd file.

**server(required)**

ICM server from which the Web services will extract the item types.

**b(optional)**

See the explanation to the URL for downloading the cmbmessages.xsd file. If you want to regenerate the proxy classes using new WSDL and schema files downloaded from DB2 Content Manager 8.4 Web services server and do not want to change the other parts in their applications that use the proxy classes, they can download the itemtype.xsd file by using the parameter with the value true. If the parameter is not set or its value is not true, the downloaded itemtype.xsd file generates incompatible proxy classes for those client applications mentioned above.

**credential(optional)**

True indicates that the HTTP request to the URL has the Single Sign-On (SSO) credential.



## Authenticating to ICM datastore

If the **credential** parameter is in the query string and the value of the parameter is true, the Web services server always looks up a cookie named LtpaToken in the HTTP request and tries to extract a credential string. If the credential string exists, Web services server uses the credential string to connect to the ICM server and then extract the specified item types.

If the **credential** parameter does not exist or the value of the parameter is not true, the Web services server sends a HTTP response back to the HTTP client with a status code 401 (authentication failed) and WWW-Authenticate header. The HTTP client must supply user ID and password, and encode them into the authorization header of the next HTTP request to the Web services server. After receiving the HTTP request, the Web services server decodes the user ID and password from the authorization header and use them to connect to the ICM server and then extract the specified item types.

**Important:** The **itemtype** parameter in the URLs for downloading the cmbmessages.xsd and itemtype.xsd files must be exactly same. Otherwise the cmbmessages.xsd file cannot correctly refer to the XML schema declarations for item types in the itemtype.xsd file. Consequently, the proxy classes cannot be generated.

If the **itemtype** parameter is not in the URL for downloading the cmbmessages.xsd file, the cmbmessages.xsd file does not include the itemtype.xsd file. In this case, the CMWebService.wsdl, cmbmessages.xsd, and cmbdatamodel.xsd files are sufficient to create the proxy classes for a client application that is not bound with specific item types. The application can work with any item types. But the application developers have to write their own program for the marshalling or de-marshalling any XML representation for the item types in the input and output messages.

### Generating the WSDL file for the CMWebService Web service from system administration client:

Exporting item types to WSDL is still supported in DB2 Content Manager Version 8.4 system administration client. It generates the WSDL and the supporting schema files for a group of item types.

To generate the WSDL and its supporting schema files for a group of item types by using the system administration client, follow these steps:

1. In the system administration client, select the item types.
2. Right-click and select **Export to WSDL file**.
3. Type the default name, CMWebServiceWSDL.zip. The default directory is *IBMCMROOT*/config, where *IBMCMROOT* is the home directory for DB2 Content Manager V8.4.
4. Change the default directory to save the ZIP file.
5. Click **Save**. The CMWebServiceWSDL.zip file will be saved to the local hard drive.

The CMWebServiceWSDL.zip files contain these files:

- CMWebService.wsdl
- cmbmessages.xsd
- cmbdatamodel.xsd

- itemtype.xsd

The limitations of using the system administration client to export item types to WSDL are:

- The cmbmessages.xsd file is always bound with item types. It is impossible to generate a cmbmessages.xsd file without the references to any item type.
- The backward compatibility supported by the query string parameter **b** in the URLs for downloading the cmbmessages.xsd and itemtype.xsd files is not available. The WSDL and XML schema files in CMWebServiceWSDL.zip exported from the system administration client generate the proxy classes incompatible with the Web services client applications connected to CMWebService which were developed with DB2 Content Manager Version 8.3 Fix Pack 4 or earlier.

#### **Related reference**

“Using the forceItemNamespace property when developing client applications that use the Web services” on page 438

The forceItemNamespace property allows a namespace to be set for item type schema information.

#### **Generating the WSDL file for the CMWebService Web service from command line:**

You can use the command-line Java tool, CMBGetSpecificWSDL to export the CMWebServiceWSDL.zip for CMWebService. For more information, see WEB SERVICES UTILITY - WSDL GENERATION TOOL . Using command-line Java tool is an alternative to generating the WSDL file from system administration client, but it has command line argument to support the backward compatibility for the Web services client applications using CMWebService which were developed with DB2 Content Manager Version 8.3 Fix Pack 4 or earlier. However the Java tool still cannot generate cmbmessages.xsd file that is not bound with special item types.

#### **Getting started with the CMWebService in a .NET environment**

When working with the Web services in a .NET environment, you can either generate the XML by using the Microsoft SOAP Toolkit to create low level routines, or you can use the WSDL tool to generate proxy classes.

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 send the document to the Web service URL and returns the reply document as part of the SOAP message. This is a low-level interaction with the Web services. This type of interaction allows the most flexibility because the Web services interface does not change, even if the XML schema changes.

The disadvantage to using the Web service this way is that you must generate XML messages and deal with low-level APIs for sending and receiving SOAP messages. In this case, the WSDL is used only for specifying the endpoint URL of the Web service.

To create XML and SOAP requests, you can write a Web services client that uses a WSDL utility. A WSDL utility can automatically process your XML and SOAP requests and responses by representing them as proxy classes. Microsoft provides a WSDL utility that can represent XML documents as C# proxy classes.

To write an application that uses the classes generated by Microsoft .NET's WSDL utility, follow these steps:

1. Ensure that the following software is installed:

**.NET 1.1 client**

- .NET Framework SDK Version 1.1 from the Microsoft
- .NET Web Service Enhancements SP1 from the Microsoft
- .NET Framework SDK, Version 1, Release 1, Hotfix 892202 from the Microsoft

**.NET 2.0 client**

- .NET Framework SDK version 2.0 or 3.0
- Web Services Enhancements SDK 3.0

2. Create C# proxy classes from the WSDL and schema files.
3. Using the C# samples for guidance, program your Web services application client by using the C# proxy classes.

**Important:**

When you use the C# samples, it is assumed that the server is local. However, the server and the Web services samples can be installed on different machines, in which case you must modify the server URL accordingly.

In addition, the proxy classes that are shipped with the samples might not match the sample data stored in First Steps. This can happen if the sample data was imported with a fix pack level that is different from the level in which the Web services are running. In these cases, the samples will not work as shipped, and you must generate a new WSDL and new proxy classes.

**Related concepts**

"Programming Web services requests in a .NET environment" on page 424

**Related tasks**

"Generating the WSDL file for the CMWebService Web service from system administration client" on page 420

Exporting item types to WSDL is still supported in DB2 Content Manager Version 8.4 system administration client. It generates the WSDL and the supporting schema files for a group of item types.

"Generating proxy classes for .NET 1.1 client"

After you have a WSDL, you must turn that WSDL into usable proxy classes for WSDL.

"Generating proxy classes for .NET 2.0 client" on page 423

**Generating proxy classes for .NET 1.1 client**

After you have a WSDL, you must turn that WSDL into usable proxy classes for WSDL.

To generate the proxy classes, you must install the following software:

- .NET framework SDK, Version 1, Release 1
- .NET Web Services Enhancements, Version 1, Service Pack 1
- .NET framework SDK, Version 1, Release 1, Hotfix 892202 is installed. Contact Microsoft for this hotfix.

To generate proxy classes, follow these steps:

1. Extract the CMWebService.zip file into a temporary directory. The following files must be extracted:

- CMWebService.wsdl
- cmbmessages.xsd
- cmdatamodel.xsd
- itemtype.xsd

2. Open a command-line window and type the following command on one line:

```
C:\VisualStudio2003\SDK\v1.1\Bin\wsdl.exe /language:CS
/out:CMWebService.cs file:///C:\CMWebServiceWSDL\CMWebService.wsdl
```

You can do this step in Visual Studio .NET 2003 by adding a Web reference to file:///C:\CMWebServiceWSDL\CMWebService.wsdl.

The command assumes that .NET SDK v1.1 is installed under C:\VisualStudio2003\SDK\v1.1 and the CMWebService.zip file is extracted in C:\CMWebServiceWSDL. The command generates a C# class file, CMWebService.cs

You might get warning messages when you are executing the command, but you can ignore them. The proxy class file is generated correctly.

3. Modify the C# proxy class. The proxy class CMWebService.cs is derived from System.Web.Services.Protocols.SoapHttpClientProtocol by default. To use the functionality of WSE 1.0 SDK, change the base class to WebServicesClientProtocol:

- Open the CMWebService.cs file in an editor. Add an using namespace clause. using Microsoft.Web.Services;
- Replace the class declaration code in the original example with the following modified example:

**Original**

```
public class CMWebService :
 System.Web.Services.Protocols.SoapHttpClientProtocol {
```

**Modified**

```
public class CMWebService :
 Microsoft.Web.Services.WebServicesClientProtocol {
```

- Save the CMWebService.cs file.

## Generating proxy classes for .NET 2.0 client

To generate the proxy classes, you must install the following software:

- .NET framework SDK, Version 2.0
- .NET Web Services Enhancements, Version 3.0

To generate proxy classes, follow these steps:

1. Extract CMWebService.zip into a temporary directory. The following four files must be extracted:

- CMWebService.wsdl
- cmbmessages.xsd
- cmdatamodel.xsd
- itemtype.xsd

2. Enter the following command: C:\Program Files\Microsoft WSE\v3.0\Tools\WseWsd13.exe /language:CS /out:CMWebService.cs /type:webClient file:///C:\CMWebServiceWSDL\CMWebService.wsdl

You can do this step in Visual Studio 2005 by adding a Web reference to file:///C:\CMWebServiceWSDL\CMWebService.wsdl.

### Related reference

“Using the `forceItemNamespace` property when developing client applications that use the Web services” on page 438  
The `forceItemNamespace` property allows a namespace to be set for item type schema information.

## Programming Web services requests in a .NET environment

You can use the `CMWebServiceClient.cs` sample to help you learn how to create a Web services request. The `CMWebServiceClient.cs` sample relies on the First Steps data, so ensure that you installed the First Steps component.

You can do the following tasks to develop an application in a .NET environment:

### Create 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;
```

### Authenticate 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 428.

### Create a new instance of an item

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 429

### Wrapping the XML request

The `CMWebServiceClient.cs` sample specifies direct XML requests through the WSDL generation utility, for example:

```
elementRequest request = new RetrieveItemRequest();
request.AuthenticationData = authData;
request.attribute = elementattribute.att_value;
elementReply reply=webservice.element(request);
```

For details about the XML request syntax, see Programming run-time operations through the XML JavaBeans.

### Attach binary content (if applicable)

You can use one of three 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 431.

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;
 }
}

```

## Integrating CMBGenericWebService

By using the CMBGenericWebService, 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 main steps that you perform are as follows:

1. Get the WSDL file from the following URL: `http://hostname/CMBGenericWebService/services/CMBGenericWebService/wsdl/CMBGenericWebService.wsdl`
2. Generate proxy classes for the WSDL.
3. Write your application program to use the proxy classes to communicate with the CMBGenericWebService. The URL for the CMBGenericWebService is `http://hostname/CMBGenericWebService/services/CMBGenericWebService`.

The Java samples with WebSphere Application Server 6 are in `IBMCMROOT/samples/webservices/GenericWebService/java/WAS6/base`.

For details, see “Customizing Java applications to work with CMBGenericWebService.”

### Getting the WSDL file for the CMBGenericWebService Web service

The WSDL file is a key item when working with the Web services.

The WSDL file for the CMBGenericWebService is static and is installed on the Web server when the Generic Web service is installed. To view the WSDL file, you must have the Web services installed on your system.

To get the WSDL file for CMBGenericWebService, follow these steps:

1. Type the following URL into a browser: `http://localhost/CMBGenericWebService/services/CMBGenericWebService/wsdl/CMBGenericWebService.wsdl`.
2. Type the correct host name and port number for your system.
3. To save the WSDL, in the browser menu, click **File** → **Save As** and select the directory where you want to save the WSDL file. Click **Save**.

You can now view the WSDL in a browser.

### Customizing Java applications to work with CMBGenericWebService

You can customize your Java application to create the XML requests to send to the Web services.

You can use any JAX-RPC based client toolkit to generate the classes that invoke the Web services and pass the parameters back and forth to the Web services. WebSphere Application Server Version 6 provides a client tool called WSDL2Java that you can use to generate the client classes for the Web services. Because the

WSDL file does not define the syntax of the XML documents, the interface of the Web services does not change if the XML schema for the request changes.

To write an application in the Java environment, follow these steps:

1. Install the WebSphere Application Server Version 6.1.
2. Use WebSphere Application Server's WSDL2Java tool to generate Java proxy classes from the WSDL for the CMBGenericWebService Web service.
3. Using the Java samples as guidance, program your Web services application by using the Java proxy classes.

#### **Related concepts**

"Programming Web services requests in a Java environment"

#### **Related tasks**

"Getting the WSDL file for the CMBGenericWebService Web service" on page 425

The WSDL file is a key item when working with the Web services.

**Programming Web services requests in a Java environment:** After you set up your Java Web services environment, you can use GenericWebServiceSample.java as a basis for creating a Web services request. In summary, a Java application contains the following code:

#### **Create a Web service object**

To instantiate a Web services object, use the following example:

```
CMBGenericWebServiceService cs =
 new CMBGenericWebServiceServiceLocator();
cmbservice = cs.getCMBGenericWebService();
```

#### **Authenticate the Web services request**

For security, you must create an authentication object in each request.

#### **Create a new instance of an item (if applicable)**

The GenericWebServiceSample.java sample creates an instance of an XYZ\_InsPolicy item.

#### **Wrap the XML request**

The GenericWebServiceSample.java sample passes parameters into predefined message templates to create XML requests. For example:

```
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 the file, 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 details about the XML request syntax, see “Programming run-time operations through the XML JavaBeans” on page 352.

#### **Create a DOM from an XML string**

Building a DOM (Document Object Model) 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)));
```

#### **Attach binary content (if applicable)**

The Web services client using CMBGenericWebService endpoint can upload or download binary data to CMBGenericWebService endpoint by using MIME as the attachment format.

#### **Parse the Web services response**

The GenericWebServiceSample.java sample contains various methods for validating XML responses and reporting errors.

The following example wraps an XML response to retrieve the XYZ insurance policy (and a URL for its resource part) by using 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;
}

```

#### Related concepts

“Web services samples” on page 437

A set of thoroughly commented Web services samples are provided to help you work with the Web services.

## Authenticating Web services requests for security

Every time 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 change an insurance policy but has only view privileges, the user cannot change the policy.

**Important:** By default, the user ID and password passed in the Web services request are not encrypted. If all of the Web services requests are being processed within the firewall, unencrypted IDs and passwords might not be a problem. 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.

The following specific C# example returns an initialized AuthenticationData object where *server* represents the DB2 Content Manager library server database name (for example, *cmdb21s*), *userid* represents your DB2 Content Manager user ID (for example, *icadmin*), 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 Java sample creates an authentication object where *dstype* represents the data source type (for example, ICM), *server* represents the DB2 Content Manager library server database name (for example, *cmdb21s*), *username* represents your DB2 Content Manager user ID (for example, *icadmin*), and *password* represents your login 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

This example creates an instance of an XYZ\_InsPolicy item.

#### C# sample

The following C# code snippet proxy classes that you use to construct the XML representation of an item of item type XYZ\_Policy, which will be a part of CreateItemRequest.

```
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_InsrdLName = 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

Using the templates provided in the SampleMessageTemplate.java Web services Java sample, the following Java code snippet creates XML that represents an item of item type XYZ\_Policy.

```

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()});
}

```

### Related concepts

“Items” on page 64

An *item* is the basic entity managed by the library server.

## Attaching binary content parts to items in Web services

The DB2 Content Manager Web services samples rely on standard binary message formats to send attachments (such as base parts, annotations, or notelogs) inside XML requests.

The following standard binary message formats are used by the DB2 Content Manager Web services:

### Message Translation Optimization Mechanism (MTOM)

MTOM is a W3C standard method for sending binary data to and from Web services. It uses XOP (XML-binary Optimized Packaging) to transmit binary data and is intended to replace both MIME and DIME attachments. Only the CMWebService supports MTOM types.

The following example creates a MTOM attachment object to hold content parts (in the resources array) within the DB2 Content Manager items. The code can be found in CMWebServiceClientWSE30 sample project.

#### C# sample

To use an MTOM attachment, the binary data must be loaded into a XML element named MTOMAttachment.

```

//<summary> Sets up an array of MTOMAttachments.
//The MTOMAttachments are used to send the content
// of the documents parts to the server in the soap message
//</summary>
//<param name="resources"> array containing information
//about the document part to be associated
//with the policy </param>
//<returns> new array of MTOMAttachments</returns>

```

```

MTOMAttachment[] setupAttachments(string[,] resources)
{
 //Create MTOMAttachment objects for all the resources
 //passed in the resources array provides
 // the information for the document parts to be
 //sent to the server
 MTOMAttachment[] attachments = new
 MTOMAttachment[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 MTOMAttachment();
 attachments[i].ID = resources[i, 0];
 attachments[i].MimeType = resources[i, 1];
 attachments[i].Value = File.ReadAllBytes(resources[i, 2]);
 }
 return attachments;
}

```

Then the MTOMAttachment object must be referred to as a child element of request XML message mtomRef.

```

//<summary>
//Creates a new policy on the Content Manager server.
//</summary>
//<param name="authData">authentication data to be used for
//connecting to the datastore
//</param>
//<param name="policy">insurance policy to be created</param>
//param name="attachments">array of MTOMAttachment representing
//the content of the document parts to be created for
//the policy on the server
//</param>
//<returns> the PID URI of the created policy
If an error occurs, null is returned. </returns>
public string createPolicy(AuthenticationData authData,
 XYZ_InsPolicy policy, MTOMAttachment[] attachments)
{ //Create a CreateItemRequest object to set up the request for
 //creating a XYZ_InsPolicy
 CreateItemRequest request = new CreateItemRequest();
 request.AuthenticationData = authData;
 //set the authentication information for the create request
 request.Item = new CreateItemRequestItem();
 //Because we are creating one item, we have an array
 //of one CreateItemRequestItem object
 //CreateItemRequestItem has an ItemXML object which
 //points to the in-memory instance of the item to be created.
 //In this case the item is an instance of XYZ_InsPolicyrequest.
 Item.ItemXML = new ItemXML();
 request.Item.ItemXML.XYZ_InsPolicy = policy;
 // Add MTOM attachments
 if (attachments != null)
 {
 request.mtomRef = new MTOMAttachment[attachments.Length];
 for (int i = 0; i < attachments.Length; i++)
 {
 request.mtomRef[i] = attachments[i];
 }
 }
 CreateItemReply reply = webservice.CreateItem(request);
 //invoke the Content Manager web service
 //with the CreateItem operation
 // check to see if the operation was successful or not
 if (reply.RequestStatus.success == true)
 { // return the PID URI of the created item
 return reply.Item.URI;
 }
 else
 {
 Console.WriteLine("Create failed.");
 }
}

```

```

 displayErrorInfo(reply.RequestStatus.ErrorData);
 return null;
 }
}

```

### Multipurpose Internet Mail Extensions (MIME)

The MIME standard messaging protocol can identify binary files and types through the Content-ID or Content-Location headers in a SOAP envelope. The CMBGenericWebService and CMWebService support MIME types. Both the 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 size of a file ahead of time in the header fields to save the time of

calculating it. This protocol is the only format supported for handling digitally signed or large binary files in a .NET environment. Only the CMWebService supports DIME types. 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;
}
```

## Configuring single sign-on

Single sign-on is used by Web applications to ensure that the user of the application only needs to supply the user ID and password one time.

To set up single sign-on (SSO) for the Web services, you must configure SSO for both DB2 Content Manager and WebSphere Application Server. Then, you can connect to DB2 Content Manager Version 8 server through Web services using the credentials generated by WebSphere Application Server in each request instead of supplying the user ID and password in every request. For instructions on configuring for SSO, see *Configuring the eClient for single sign-on (SSO)*

To verify that the SSO is successfully configured, restart Web services application and complete the following steps.

1. Access the URL <http://hostname/CMBSpecificWebService/index.html> in a Web browser.
2. Enter the user ID and password defined during configuring WebSphere Application Server security.

If the user ID and password are correct, a text message DB2 Content Manager Specific Web Service will be printed in the Web browser.

### LTPA tokens and single sign-on

Single sign-on is used by Web applications to ensure that the user of the application only needs to supply the user ID and password one time.

When a user logs on to a single sign-on enabled server, single sign-on creates a Lightweight Third Party Access (LTPA) token. That LTPA token is used for authentication of that user on future requests.

The LTPA token contains the credential information used for accessing the server. The credential information is used in subsequent requests made to the Web services applications from the client machine. There are various ways to obtain the credential information, which is stored as a cookie on the client machine, and is also available in the HTTP request on subsequent trips to the server. One way is to implement a servlet on the server that can extract the credential information from incoming HTTP requests.

### Example

```
/**
 * Retrieve the LTPA token from the HttpServletRequest
 * if the SSO authentication was successful,
 * any subsequent request from the browser will
 * have an SSO cookie in the header
 *
 * @return String
 */
public String getCookie(HttpServletRequest request) {
 Cookie cookies[] = request.getCookies();
 String cookievalue = null;
 for (int i=0; i<cookies.length; i++){
 if (cookies[i]
 .getName().equalsIgnoreCase("LtpaToken")) {
 cookievalue = cookies[i].getValue(); break;
 }
 }
 return cookievalue;
}
```

When including this credential in requests to the Web service, you must first convert it to base 64 encoding, as shown in the following example:

```
String cookie = getCookie(request)
// client application does not necessarily need to run
//inside the servlet, just implemented here as an example.;
Object credential = org.apache.soap.encoding.soapenc.Base64.encode
(cookie.getBytes());
```

You can then use the credential in the Authentication Data section of a request to the Web services. When a request containing a credential is received, a search is performed in WebSphere Application Server to identify the user.

### Example

```
protected static final String AUTHENTICATION_DATA_TEMPLATE =
 "<AuthenticationData connectString=\"SCHEMA=ICADMIN\"
 configString=\"\">"
 + "<ServerDef>"
 + "<ServerType>{0}</ServerType>"
 + "<ServerName>{1}</ServerName>"
 + "</ServerDef>"
 + "<Credential>{2}</Credential>"
 + "</AuthenticationData>"
protected static final String QUERY_TEMPLATE =
 "<RunQueryRequest xmlns=\"http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema\"
 maxResults=\"0\"
 + \"version=\"latest-version(.)\" contentOption=\"{0}\" retrieveOption=\"{1}\">"
 + "{2}"
 + "<QueryCriteria>"
```



```

 + "<QueryString>{3}</QueryString>"
 + "</QueryCriteria>"
 + "</RunQueryRequest>";
 .
 .
 .
 String queryString = "/XYZ_InsPolicy[@XYZ_State LIKE \"
 String authenticationXML =
 MessageFormat.format(
 AUTHENTICATION_DATA_TEMPLATE,
 new Object[] { "ICM", "icmnlbdb", credential });
 String requestXML =
 MessageFormat.format(
 QUERY_TEMPLATE,
 new Object[] { "URL", "ITEMTREE", authenticationXML, query });

 CMBXMLResponse response = null;
 try {
 response = cmbservice.processXMLRequest(requestXML, null);
 } catch (RemoteException e) {
 e.printStackTrace();
 return null;
 }
}

```

---

## Web services performance

To improve the performance of a Web service, consider heap size, amount of data retrieved, and database connection pooling appropriately.

If the heap size is not set properly on WebSphere Application Server, you might receive exceptions. If you receive an `OutOfMemory` exception, your maximum heap size might be too low. Try increasing the value but make sure that the value for the Java Virtual Machine heap size is directly related to the amount of physical memory for the system. Never set the Java Virtual Machine heap size larger than the physical memory on the system to avoid disk I/O caused by swapping. The information includes setting Java Virtual Machine parameters for garbage collection options, such as **-Xgcpolicy:optavgpause** and generic Java Virtual Machine arguments such as **-Xnoclassgc**.

The way your client application is programmed can also contribute to the `OutOfMemory` exception. You can set a limit to the amount of data to be retrieved from the Web services server. Use the parameters in the Web services requests to set the limit.

In `RunQueryRequest`, the `maxResults` attribute can be used to limit the number of items to be returned by the request. `RunQueryRequest` not only searches for items PID that satisfy the specified query string, but also retrieves them according to the `retrieveOption` attribute. For best performance, a typical client application can implement its own pagination function. With pagination, the client can send a `RunQueryRequest` with `retrieveOption` being set to `IDONLY` to get PIDs for items that satisfy the query string, and send a `RetrieveItemRequest` to retrieve items on every page that is currently being read. This approach will reduce the response time for the end users of your client application.

**Important:** Use the `maxResults` attribute in `RunQueryRequest` to avoid retrieving an unreasonably large number of PIDs.

`RetrieveFolderItemsRequest` has `maxItems` attribute that enables you to set the limit of items to be returned when opening a folder. Client pagination is also recommended in the implementation of functions, such as folder contents.

ListWorkPackagesRequest has a different way to set the limit of workpackages to be returned. When you are defining a worklist in the system administration client, set a maximum value for **Quantity to Return** on Worklist definition window.

In addition, for high volume, concurrent Web service requests consider enabling database connection pooling on WebSphere Application Server. Recommendations are outlined in the tuning guide for enabling database connection pool for the eClient application. You can use that same information for the Web services.

For best performance, follow the suggestions that are outlined in the Performance tuning for Content Manager related to WebSphere Application Server tuning for the eClient. The instructions contained in this IBM Redbooks are also useful for configuring the Web services.

---

## Web services samples

A set of thoroughly commented Web services samples are provided to help you work with the Web services.

To get started working with the Web services samples, see the appropriate file for your environment in *IBMCMROOT/samples/webservices*:

### **GenericWebServiceSampleWAS6.readme**

Start with this file if you are working with the Web services on WebSphere Application Server Version 6.1

### **CMWebServiceClient.readme**

Start with this file if you are working with the Web services in a .NET 1.1/WSE 1.0 environment.

### **CMWebServiceClientWSE30.readme**

Start with this file if you are working with the Web services in a .NET 2.0 or 3.0/WSE 3.0 environment.

### **CMWebServiceClientJava.readme**

Start with this file if you are working with the Web services in a Apache Axis 1.4 environment.

---

## Web services troubleshooting

Troubleshooting information can help you diagnose and solve problems with the Web services.

The information in this section provides troubleshooting information to help you solve some common Web services problems.

### **Setting the cached statement in WebSphere Application Server connection pool for Web services**

If you select a large number for the cached statements in the WebSphere Application Server connection pool, your Web services application can run out of memory and stop running.

#### **Action**

When using WebSphere Application Server connection pooling, select a value that is less than 10 for the cached statements.

## Using the `forceItemNamespace` property when developing client applications that use the Web services

The `forceItemNamespace` property allows a namespace to be set for item type schema information.

In DB2 Content Manager Version 8.4, the `forceItemNamespace` property in `cmbxmllservices.properties` file is always enabled and its value is set to `http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema` by default. You should not change unless you get the backward compatibility issues for the Web services client applications using `CMWebService` that were developed with DB2 Content Manager Version 8.3.

In DB2 Content Manager Version 8.3, the `cmbxmllservices.properties` file does not have the `forceItemNamespace` property by default. However, you can add `forceItemNamespace=http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema` line manually to `cmbxmllservices.properties` file.

If you have developed a Web services client application connected to DB2 Content Manager Version 8.3 specific Web services (`CMWebService`), the application does not retrieve items from DB2 Content Manager Version 8.4 Web services server when the `forceItemNamespace` property was not in `cmbxmllservices.properties` file or the property is commented out

To solve this problem, you can choose any of the two choices:

- Comment out the line `forceItemNamespace=http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema` in `cmbxmllservices.properties` file and restart DB2 Content Manager Version 8.4 Web services server. You do not have to change your application.
- Download or export the WSLD and XML schema files for DB2 Content Manager Version 8.4 `CMWebService` and regenerate the proxy classes from the WSLD and XML schema files. The code of other parts of your application does not need to be changed. But, you must compile and build the whole application again.

**Important:** .NET 2.0/WSE 3.0 client can work only with Content Manager Web Services server with `forceItemNamespace` property being enabled. The `forceItemNamespace` property must be enabled when generating Specific Web Service WSDL and schema files for the .NET 2.0/WSE 3.0 client.

## Update Web services server configuration to refer to the 32-bit libraries when using a DB2 64-bit instance

The DB2 Content Manager APIs require the 32-bit libraries. The Web services also require the same libraries.

### Symptom

You see the error: `DGL0394A: Error in DriverManager.getConnection; No suitable driver` when you send requests through DB2 Content Manager Web services when using a DB2 64-bit instance.

## Possible cause

When installing DB2 Content Manager Web services on a system with a DB2 64-bit instance, the default libraries set in the Web services server configuration in the WebSphere Application Server are 64-bit versions.

## Action

Update the Web services server configuration to refer to the 32-bit libraries by following these steps:

1. Ensure that Fix Pack 1 is installed.
2. Update the LIBPATH property used by the server configuration to contain the location of the 32-bit libraries:
  - a. Open the WebSphere Application Server administrative console.
  - b. Click **Application Servers** → **cmwebsvc** → **Process Definition Environment Entries**.
  - c. On AIX, click the LIBPATH variable so that you can edit it. On Solaris, click LD\_LIBRARY\_PATH. This variable contains one or more paths to various library locations in the file system.
  - d. The DB2 UDB libraries are found in DB2instancehome/lib where *DBinstancehome* is /home/db2inst1/sqllib by default.
  - e. Change the *lib* directory to *lib32*.
  - f. Click **OK** to save the new value.
  - g. Save the changes to the master configuration.
3. Restart the cmwebsvc server.



---

## Working with the Java document viewer toolkit

The document viewer toolkit is a set of Java™ APIs that facilitate displaying, manipulating and annotating documents.

This toolkit includes a Swing-based graphical user interface as well as non-visual classes that provide document conversion and rendering capabilities. There are many capabilities for customization and extension, allowing you to tailor the viewer to fit into specialized Content Manager applications.

---

### Java viewer toolkit overview

The Java™ viewer toolkit supports viewing, annotation, and manipulation of a variety of document formats, primarily: TIFF, MO:DCA (containing IOCA, PTOCA), GIF, JPEG, PCX, Bitmap, and text documents and Microsoft® Office documents using a bridge to the Stellent Outside-In toolkit

You can extend the viewer to support other document formats, by writing document engines that are classes which provide document format specific functions.

The Java viewer toolkit provides the following capabilities:

- Multiple-document interface with several different layouts including tabbed and side-by-side layouts. Side-by-side viewing enables two different views of a single document or two different documents. The views can be configured horizontally or vertically and edited independently.
- Thumbnails view of the pages of a document: Clicking on a thumbnail of a page to moves to that page. The currently visible area on the page is indicated 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.
- Configurable, dockable toolbars
- Customizable pop-up menus and action objects for all menu items (which can be used to build a menu bar)
- Rotating, zooming, inverting, and enhancing pages
- Image contrast adjustment and balance of pages in the Java viewer. The Balance window has slider controls and text fields to adjust the brightness and contrast.
- Page and document navigation: First, next, previous, last page, and jump to a page.
- Fit to width, height, and to window
- Printing, including selection of pages to print and the ability to add custom options to control the printing of additional information. Customization capabilities also allow an application to provide a watermark on printed pages.
- Creating and editing graphical annotations. The annotation types provided include a box, circle, line, arrow, text, highlight, sticky note, pen and stamp. Stamp annotations can be either text or images.
- Page manipulations, including copying, reordering, rotating, and deleting pages in image-paged document formats and saving the changed document.
- Multiple selection of pages in the thumbnails, useful for rotation and other manipulations of the pages.

- Copying and pasting of annotations and pages within a document and between documents.
- Copying of a selected area of a page to the clipboard (for pasting into another application).
- Undo and redo for operations that modify the document, such as editing annotations and page manipulation
- Page filtering: This feature allows an application to display a subset of the pages of a document. This capability is provided through the programming interface only.
- Ability to export a document to the file system. The document can include annotations.
- User controllable preferences, which are saved locally and are used in subsequent viewer sessions. Preferences include document settings for zoom and rotation, viewer settings for size and visibility of thumbnails, location and visibility of toolbars, and annotation settings for each annotation type (for example, border and fill color). Classes are provided to add custom preferences.

Figure 32 is an example of a generic document viewer that uses all of the default settings.

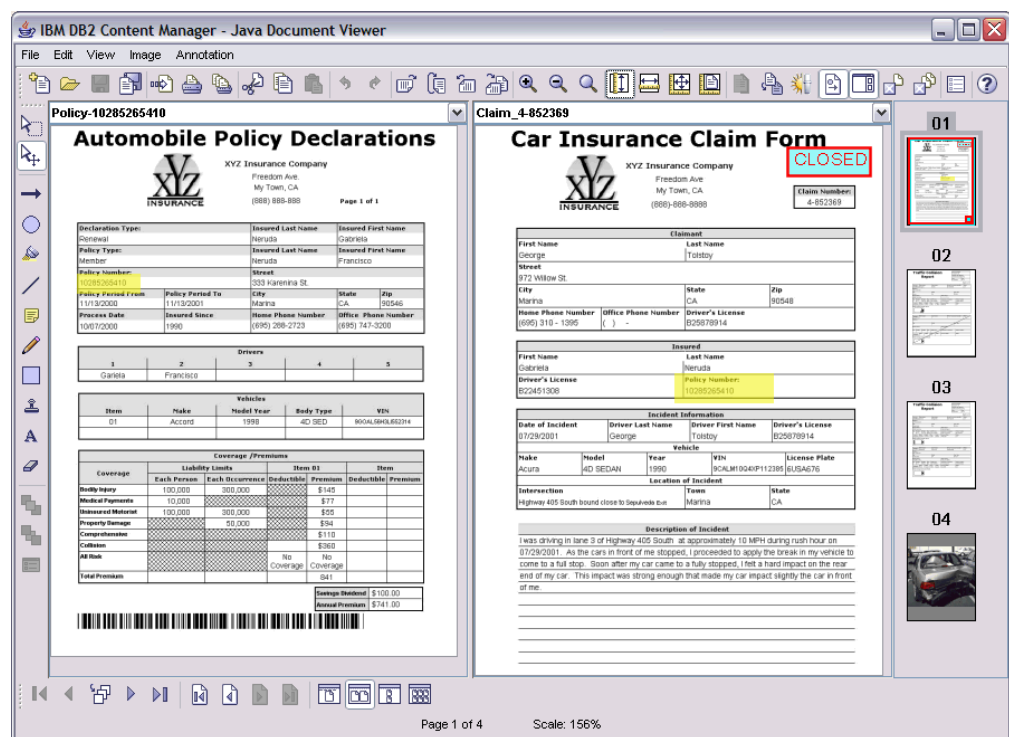


Figure 32. Generic document viewer

### Related concepts

“Working with the page manipulation functions” on page 461

The Java viewer toolkit provides capabilities for editing image-paged documents at the page level.

### Related information

 Working with page manipulation functions

## Viewer architecture

The Java viewer toolkit contains a Document Viewer bean and a Document Services bean.

The beans provide a mechanism for integrating the viewer into applications based on DB2 Information Integrator for Content JavaBeans.

The toolkit also contains the Generic Doc Viewer, Streaming Doc Services, and Annotation Services classes. With these classes, you can 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.

The Streaming Doc Services classes manage 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 Microsoft 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.

With Annotation Services classes, you can manipulate annotations. The annotation engine parses only DB2 Content Manager specific annotation formats. However, you can write additional annotation engines to handle other annotation formats. The following diagram illustrates the components that make up the Java Viewer Toolkit:

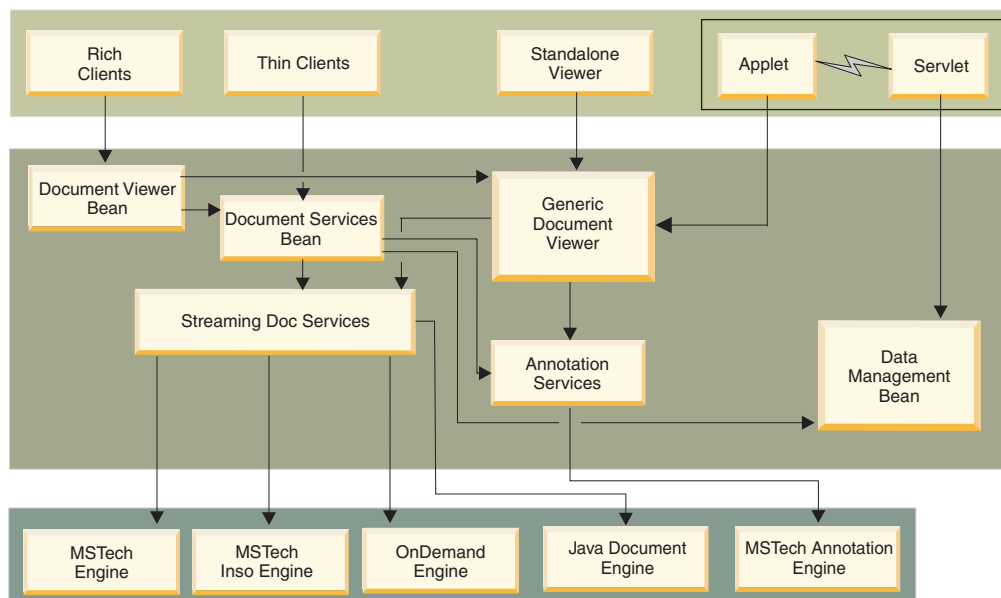


Figure 33. The components in the Java viewer toolkit

## The document engines

Document engines are used to process documents. You can use the document engines to parse documents, render pages, and manipulate the pages of a document. A document engine can work with a set of document types, like TIFF and JPEG.



The Java viewer toolkit provides the following document engines:

**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.

**DB2 Content Manager OnDemand Engine (CMBODDocumentEngine)**

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 represented by URL's to HTML with a 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, which can restrict use of the toolkit on some operating systems. The MS-Tech Document Engine and Java Document Engine are pure Java, and can be used on most operating systems. The other engines contain operating system specific logic that restricts their use to Windows.

**Related concepts**

"Supported document types and conversions in Java viewer toolkit" on page 459

The following tables summarize the document types and conversions supported by default (using the supplied conversion engines).

**Related information**



Supported document types and conversions in Java viewer toolkit

## The annotations engine

The Java viewer toolkit provides the MS-Tech annotation engine to retrieve and update DB2 Content Manager annotations.

The MS-Tech annotation engine supports DB2 Content Manager Version 8.2, Version 8.3, and Version 8.4, IBM Content Manager Version 8.1, Content Manager Version 7, and VisualInfo for AS/400 annotations.

## Example viewer architectures

To understand different ways that you can use the Java viewer toolkit, review the five example architectural designs that use the Java viewer toolkit and beans in this section.

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 URLs. You can use a standalone viewer as an applet on a Web page or for viewing documents that you obtained through e-mail.

Figure 34 illustrates a standalone viewer architecture.

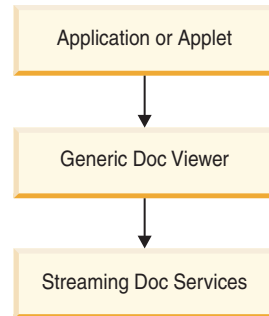


Figure 34. Standalone viewer

## Java application

To create a production Java application use the DB2 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 might have operating system dependencies. Figure 35 illustrates the architecture that you can use to build a Java application.

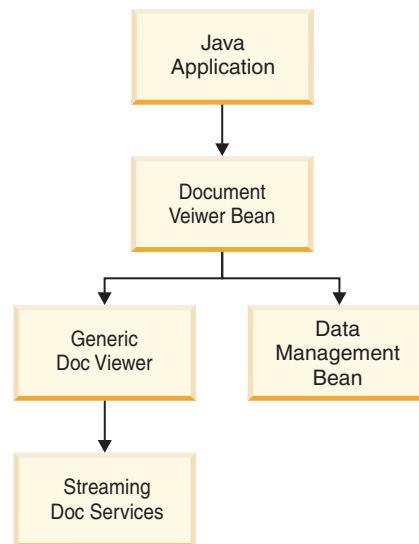


Figure 35. Java application

## 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 36 illustrates a thin client architecture.

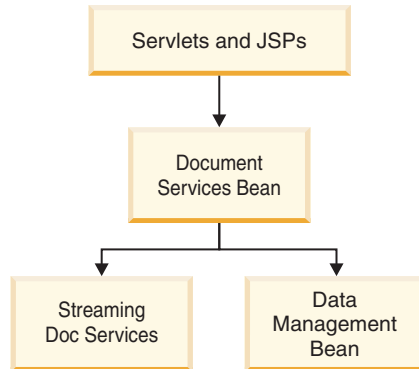


Figure 36. Thin client

### Applet and servlet

A Web-based application can also provide document viewing and annotation editing capabilities using an applet and 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 37 illustrates an applet or servlet architecture.

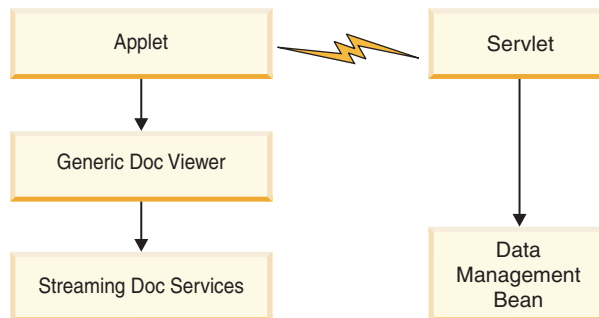


Figure 37. Applet and servlet

### Dual-mode with applet or servlet

You can use a variation of the examples provided for Web-based viewing applications. Use CMBDocumentServices on both server and in the applet.

This approach is useful when documents are not rendered in the applet but are converted on the server, which 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 Windows 2000 and the applet is running on Linux, 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 for converted documents might be slower than for locally rendered documents.

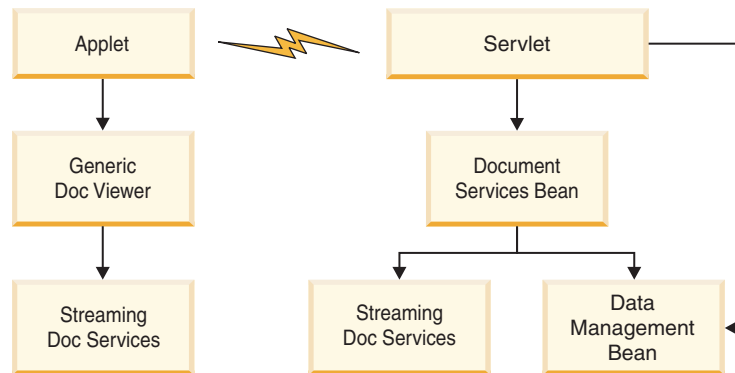


Figure 38. Dual-mode viewer

## Creating a document viewer

You can programmatically use the Java viewer toolkit classes and APIs to build your own application to view the documents.

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

This is an example of how to build a document viewer that is a standalone application.

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 each of the `CMBStreamingDocServices` and `CMBAnnotationServices` class objects by implementing the following callback classes:

- `CMBStreamingDocServicesCallbacks` abstract class
- `CMBAnnotationServicesCallbacks` abstract class

To create a standalone viewer application or applet, complete the following:

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.
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`.
  - a. Initialize it with the `CMBStreamingDocServices`, `CMBAnnotationServices`, and the configuration properties object.
  - b. 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`.

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. Specify the engines and their order using the engine property file, called `cmbviewerengine.properties`. The `CMBStreamingDocServices` object relies on document engine classes to handle the manipulation of images. The `cmbviewerengine.properties` is contained in `cmbview81.jar`. You can also do this programmatically as demonstrated in the following code snippet:

```
// 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);
```

## Example

The following code shows how to create 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);
```

### Related concepts

“Customizing the viewer” on page 450

A viewer, provided by CMBGenericDocViewer, is included with the product. You can customize the functionality provided by the viewer through the configuration properties file.

### Related information

 Customizing the viewer

## 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. Document services is the non-visual layer of the viewer and provides a model for working with the document, visually or non-visually. This returns an instance of `CMBDocument`.
2. Call `loadAnnotationSet` in `CMBAnnotationServices` to load any annotations into annotation services. Annotation services provides a model onto the annotation set for a document and allows the annotations to be manipulated both visually and non-visually. This returns an instance of `CMBAnnotationSet`.
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);

```

## 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.

The configuration properties 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.

You can customize the default configuration object, `CMBViewerConfiguration.properties`, located in the `cmbview81.jar` file, or you can create a new configuration object.

## Providing configProperties object to the viewer

The following code snippet demonstrates how to provide a custom `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);
customProperties = 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);

```

## Customizing the toolbar

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 keyword `separator`.

```

//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.



## Adding a custom tool

The following code snippet demonstrates how to add a custom tool:

```
open_doc.label=Open
open_doc.tooltip=Open
Document open_doc.icon=OpenDocument_normal.gif
open_doc.key=control 0
open_doc.className=TGenericDocViewer$OpenAction
```

The viewer creates the tool as a `CMBViewerAction`. You must provide the class that provides the action of your custom tool.

## Define action class for custom tool

The following code snippet demonstrates how to define the class that provides the action for your custom tool:

```
// This class handles the Open action. This is a custom action
// added to the operations toolbar. It must be static since it is
// instantiated from within CMBGenericDocViewer.
public static class OpenAction extends CMBViewerAction {
 public OpenAction(CMBGenericDocViewer viewer) {
 super(viewer);
 }
 public OpenAction(CMBGenericDocViewer viewer, String name) {
 super(viewer, name);
 }
 public OpenAction(CMBGenericDocViewer viewer, String name, Icon icon) {
 super(viewer, name, icon);
 }
 // @see java.awt.event.ActionListener#actionPerformed(ActionEvent)
 public void actionPerformed(ActionEvent e) {
 // Find the TGenericDocViewer instance(it is a parent of CMBGenericDocViewer)
 TGenericDocViewer gdvFrame = (TGenericDocViewer) getViewer()
 .getParent()
 .getParent()
 .getParent()
 .getParent();
 JFileChooser chooser = new JFileChooser(System.getProperty("user.dir"));
 chooser.setMultiSelectionEnabled(true);
 int ret = chooser.showOpenDialog(gdvFrame);
 if (ret == JFileChooser.APPROVE_OPTION) {
 File [] files = chooser.getSelectedFiles();
 for (int i = 0; i != ENTITY!* files.length; i++) {
 gdvFrame.openDocument(files[i].getAbsolutePath(),i == 0);
 }
 }
 }
}
```

## Using viewer actions in your application

The following code snippet demonstrates how to add a viewer action to the menu bar of 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);
```

## Customizing popup menus

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:

```
//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
```

## Adding submenus

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`:

```
//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
```

The following sections describe other viewer options that you can customize. 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
```

The thumbnail size may also be a factor of this width and height using `ThumbnailSize`. Example:

```

small - actual thumbnail size would be ThumbnailSize.width/2,
ThumbnailSize.height/2
medium - actual thumbnail size would be ThumbnailSize.width,
ThumbnailSize.height
large - actual thumbnail size would be ThumbnailSize.width * 2,
ThumbnailSize.height * 2
By default ThumbnailSize is medium
ThumbnailSize=medium

```

## 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
```

## Customizing viewer behavior

The code snippet shows how to customize other viewer behavior options.

```

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
#PrintMargins
#Specify the print margins for a mimetype.
#Default is 0,0,0,0 (top, left, bottom, right)
#printmargins.image/tiff=1,1,1,1

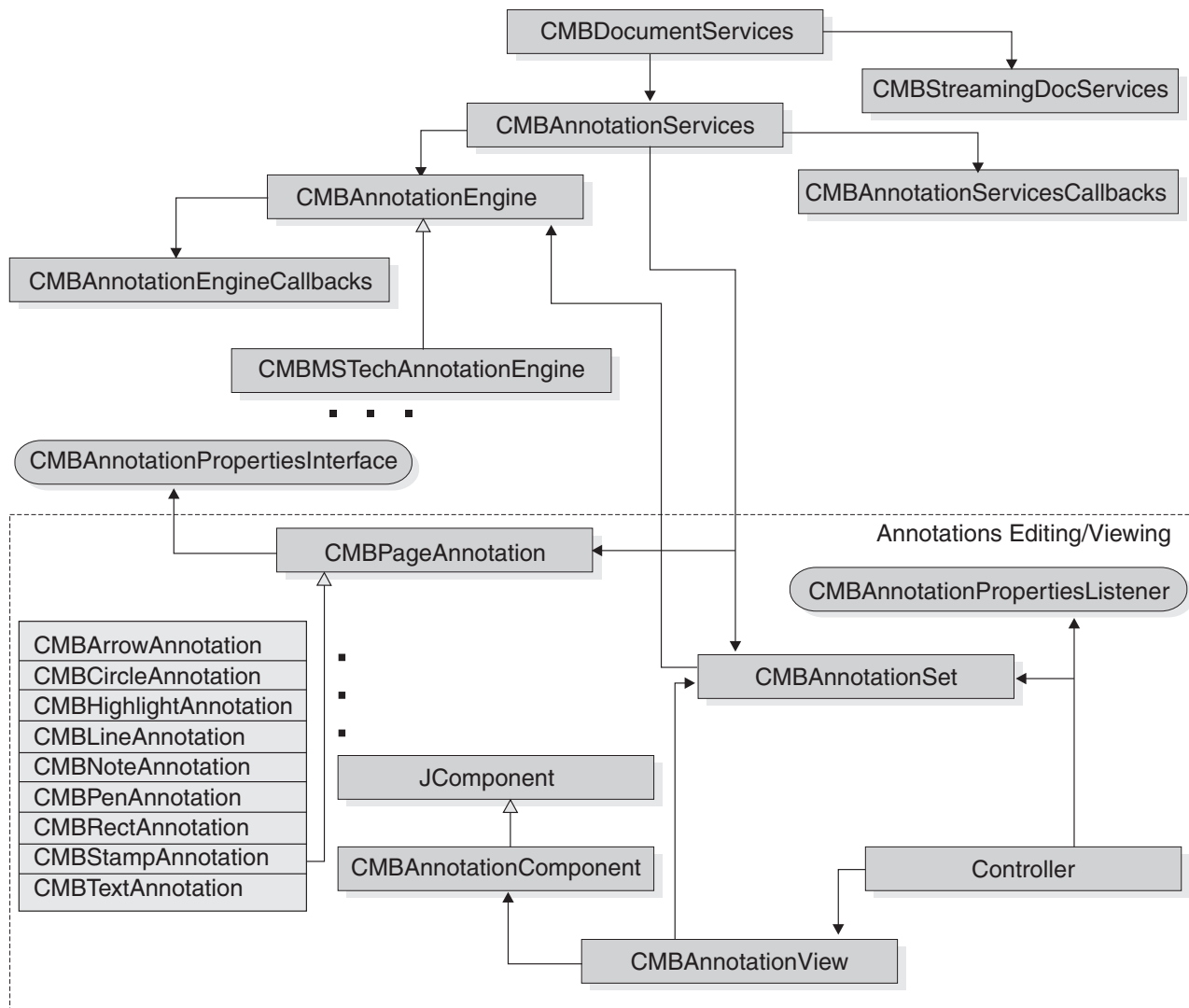
```

---

## Working with the annotation services

The DB2 Information Integrator for Content 8.4 Java viewer toolkit provides capabilities for rendering and converting document annotations.

Similar to the document services, pluggable annotation engines provide additional facilities that you can use in your applications to interpret different types of annotations. Figure 39 on page 456 shows the annotation services class diagram.



## Using annotation services interfaces

CMBAnnotationServices provides the main interfaces used in the Java viewer toolkit.

The annotation services enables you to load, manipulate, and save annotation objects using the annotation services, independently of backends. To work with annotations, you need to pass in annotation data as a stream and plug in a suitable annotation engine, which converts the annotation objects to CMBPageAnnotation instances. You can then manipulate and edit the annotations and save the annotations back into the original backend in the original format.

Figure 40 on page 457 illustrates how the generic document viewer and document services and annotation services fit together. Both document services and annotation services are used by the generic doc viewer. The callbacks for both, however, are provided by the application embedding the viewer (which is the CMBDocumentViewer visual bean in Figure 40 on page 457). Document services uses CMBDocument and CMBPage instances to represent documents and pages of

documents. Annotation services uses `CMBAnnotationSet` and `CMBPageAnnotation` (and its subclasses) to represent the annotations on a document and each individual annotation.

**Remember:** There is no direct connection between `CMBDocument` and `CMBAnnotationSet`, or `CMBPage` and `CMBPageAnnotation`. The association is maintained within the generic doc viewer. This allows the two services to be used independently of each other.

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 DB2 Information Integrator for Content 8.4 understands only the DB2 Content Manager annotation format. This annotation format is used by Content Manager Version 7, DB2 Content Manager Version 8.1, DB2 Content Manager Version 8.2, DB2 Content Manager Version 8.3, DB2 Content Manager Version 8.4, and VisualInfo for AS/400 backend servers.

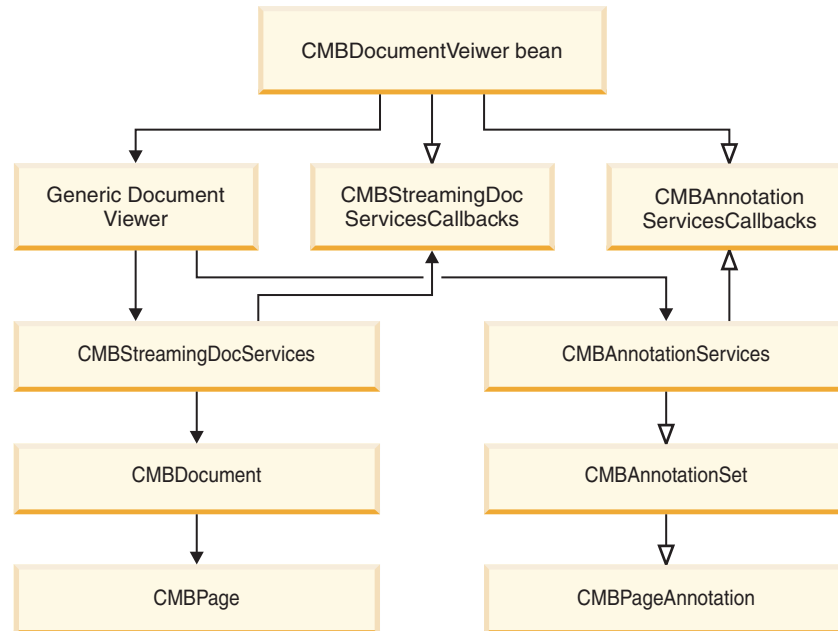


Figure 40. Annotation services and generic document viewer association

## 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, TAnnotationServices is provided in the samples directory.

## Adding a custom annotation type to your application

You can add 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.

**Remember:** This custom annotation cannot be saved because the built-in annotation engine will not support or understand it.

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.

## Supported document types and conversions in Java viewer toolkit

The following tables summarize the document types and conversions supported by default (using the supplied conversion engines).

Additional conversion capabilities can be plugged in by creating classes that extend CMBDocumentEngine and adding those to the list of engines in the engine properties passed on the constructor of CMBStreamingDocServices.

### Document types and conversions on Windows

Table 45. Document types on Windows

| MIME type                      | <name>DocumentEngine | Paginated        | With Annotations | Page Manipulation Supported | Converted To                   |
|--------------------------------|----------------------|------------------|------------------|-----------------------------|--------------------------------|
| application/afp                | CMBOD                | Yes <sup>1</sup> | No               | No                          | text/html                      |
| application/afp                | CMBOD                | No               | No               | No                          | application/afp <sup>2</sup>   |
| application/lin                | CMBOD                | No               | No               | No                          | text/plain <sup>3</sup>        |
| application/<br>ondemand line  |                      |                  |                  |                             |                                |
| image/tiff                     | CMBMSTech            | Yes              | Yes              | Yes                         | image/gif                      |
| image/tif                      |                      |                  |                  |                             | image/jpeg                     |
| image/gif                      |                      |                  |                  |                             | image/tiff                     |
| image/jpeg                     |                      |                  |                  |                             | application/<br>vnd.ibm.modcap |
| image/bmp                      |                      |                  |                  |                             | text/plain <sup>5</sup>        |
| text/plain                     |                      |                  |                  |                             |                                |
| application/<br>vnd.ibm.modcap |                      |                  |                  |                             |                                |
| image/pcx                      | CMBMSTech            | Yes              | Yes              | No                          | image/gif                      |
| image/dcx                      |                      |                  |                  |                             | image/jpeg                     |



Table 45. Document types on Windows (continued)

| MIME type                                                                                                                                                                                                                       | <name>DocumentEngine | Paginated | With Annotations | Page Manipulation Supported | Converted To            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-----------|------------------|-----------------------------|-------------------------|
| text/enriched<br>text/richtext<br>text/rtf<br>text/html<br>application/<br>vnd.lotus-1-2-3<br>application/<br>vnd.ms-excel<br>application/<br>wordperfect5.1<br>application/<br>msword<br>application/<br>vnd.lotus-<br>wordpro | CMBMSTechInso        | Yes       | Yes              | No                          | image/gif<br>image/jpeg |
| text/url                                                                                                                                                                                                                        | CMBJava              | No        | No               | No                          | text/html <sup>4</sup>  |

## Document types and conversions on AIX and Solaris

Table 46. Document types on AIX and Solaris

| MIME type                                                                                                                                                                                                                    | <name>DocumentEngine | Paginated | With Annotations | Page Manipulation Supported | Converted To                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|-----------|------------------|-----------------------------|----------------------------------------------------------------------------------------------------|
| application/afp                                                                                                                                                                                                              | CMBOD                | No        | No               | No                          | application/afp <sup>2</sup>                                                                       |
| image/tiff<br>image/tif<br>image/gif<br>image/jpeg<br>image/bmp<br>text/plain<br>application/<br>vnd.ibm.modcap                                                                                                              | CMBMSTech            | Yes       | Yes              | Yes                         | image/gif<br>image/jpeg<br>image/tiff<br>application/<br>vnd.ibm.modcap<br>text/plain <sup>5</sup> |
| image/pcx<br>image/dcx                                                                                                                                                                                                       | CMBMSTech            | Yes       | Yes              | No                          | image/gif<br>image/jpeg                                                                            |
| text/enriched<br>text/richtext<br>text/rtf text/html<br>application/<br>vnd.lotus-1-2-3<br>application/<br>vnd.ms-excel<br>application/<br>wordperfect5.1<br>application/<br>msword<br>application/<br>vnd.lotus-<br>wordpro | CMBMSTechInso        | Yes       | Yes              | No                          | image/gif<br>image/jpeg                                                                            |
| text/url                                                                                                                                                                                                                     | CMBJava              | No        | No               | No                          | text/html <sup>4</sup>                                                                             |

## Document types and conversions on Linux

Table 47. Document types on Linux

| MIME type                                                                                              | <name>DocumentEngine | Paginated | With Annotations | Page Manipulation Supported | Converted To                                                                                       |
|--------------------------------------------------------------------------------------------------------|----------------------|-----------|------------------|-----------------------------|----------------------------------------------------------------------------------------------------|
| application/afp                                                                                        | CMBOD                | No        | No               | No                          | application/afp <sup>2</sup>                                                                       |
| image/tiff image/tif<br>image/gif image/jpeg<br>image/bmp text/plain<br>application/<br>vnd.ibm.modcap | CMBMSTech            | Yes       | Yes              | Yes                         | image/gif<br>image/jpeg<br>image/tiff<br>application/<br>vnd.ibm.modcap<br>text/plain <sup>5</sup> |
| image/pcx image/dcx                                                                                    | CMBMSTech            | Yes       | Yes              | No                          | image/gif<br>image/jpeg                                                                            |
| text/url                                                                                               | CMBJava              | No        | No               | No                          | text/html <sup>4</sup>                                                                             |

### Note:

- 1 The AFP2Web toolkit performs paginated conversion to text/hml. This toolkit is not packaged with DB2 Information Integrator for Content. It is available separately. Also, the bridge to AFP2Web contained within CMBODDocumentEngine is Windows only, therefore, AFP to HTML conversion is only available on Windows using this engine.
- 2 AFP to AFP conversion also concatenates segments of large object documents.
- 3 Line data to plain text conversion also converts EBCDIC to ASCII.
- 4 text/url is a special MIME type used to indicate that a plain text document contains a URL. CMBStreamingDocServices convert the URL into a HTML document with an auto-forwarding link.
- 5 A document of any source type can be written to any of these target types only if it is possible for the target type. A multi-page document cannot be written to the image/jpeg and image/bmp types. An image document cannot be written to the plain/text type.

## 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 in the file:

```
PageManipulation=true
MultiplePageSelection=true
```

2. Update the viewer configuration to add new toolbar and menu items related to page manipulation. DB2 Content Manager provides the following actions:

| Action            | Description                                                                                                                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>new_doc</b>    | Creates a document. The document created within the viewer initially contains no pages. Saving the document creates it on the server.                                                                                                               |
| <b>save_as</b>    | Saves the document in the viewer as a new document on the server.                                                                                                                                                                                   |
| <b>export_doc</b> | 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:

| Method                                                                   | Description                                                                                                                           |
|--------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>setPageManipulationEnabled(boolean enable)</b>                        | Enables or disables all page manipulation capabilities in the viewer.                                                                 |
| <b>setPrivilege(CMBDocument document, int privilege, boolean enable)</b> | 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
 }
}
}

```

---

## Engine property settings for compression

DB2® Content Manager provides configuration options for tagged image file format (TIFF) files when you are using the document engine CMBMSTechDocumentEngine.

These configuration options enable you to specify the compression of changed pages based on the bit depth of the document. The specified compression is used to recompress only the changed pages when the document is saved. The base engine properties that are used by Streaming Doc Services, found in the cmbviewerengine properties file, include the following engine properties:

### 1-bit TIFF files

The following compressions are available with G3\_COMPRESSION as the default:

- G3\_COMPRESSION
- G4\_COMPRESSION
- HUFFMAN\_COMPRESSION
- PACKEDBITS
- UNCOMPRESSED

### 4-bit TIFF files

The following compressions are available with PACKEDBITS as the default:

- PACKEDBITS
- UNCOMPRESSED

### 8-bit TIFF files

The following compressions are available with PACKEDBITS as the default:

- PACKEDBITS
- UNCOMPRESSED

### 24-bit TIFF files

JPEG\_COMPRESSION is the only option available.

**Important:** When you use a configuration option for compression, performance is affected. You must evaluate and choose a setting that is optimal for your system.

---

## Specifying character encoding information for plain text documents

To properly decode plain text documents, the Java viewer toolkit requires that the character encoding to be provided by the application.

The CCSID information for a part can be obtained from the getCCSID method on the DKTextICM Java API or the CMBOject Java bean. This can be translated to a Java character encoding using the CMBDocumentServices.getEncodingforCCSID method. The resulting character encoding can then be provided on the CMBStreamingDocServices.loadDocument method when loading a document, and provided on the getPartEncoding callback for multi-part documents. Note that if

the encoding is not provided on loadDocument and the getPartEncoding callback is not implemented, the character encoding that will be used will be the default as defined by Java for the current locale.

For implementation details, see the following items in the *Application Programming Reference*:

**DKTextICM methods**

- getCCSID
- setCCSID

**Java Beans**

- CMBOBJECT.setCCSID
- CMBOBJECT.getCCSID
- CMBDocumentServices.getEncodingforCCSID
- CMBDocumentServices.getCCSIDforEncoding
- CMBStreamingDocServices.loadDocument
- CMBStreamingDocServicesCallbacks.getPartEncoding
- CMBDocumentEngine.loadDocument
- CMBDocumentEngineCallbacks.getPartEncoding

---

## Modifying viewer preferences

CMBGenericDocViewer contains an action and a dialog box that enable you to modify viewer preferences.

The preferences include the size of thumbnails, visual indicator for viewed documents, print window settings, and default properties of annotations. The preferences are stored in the local workstation registry. You can also export the preferences as a file and import them onto a different workstation.

The Preferences window has a tree of preference pages on the left and preferences for a selected page on the right. When you open the Preferences window for the first time, the general viewer preferences display, see Figure 41 on page 466:

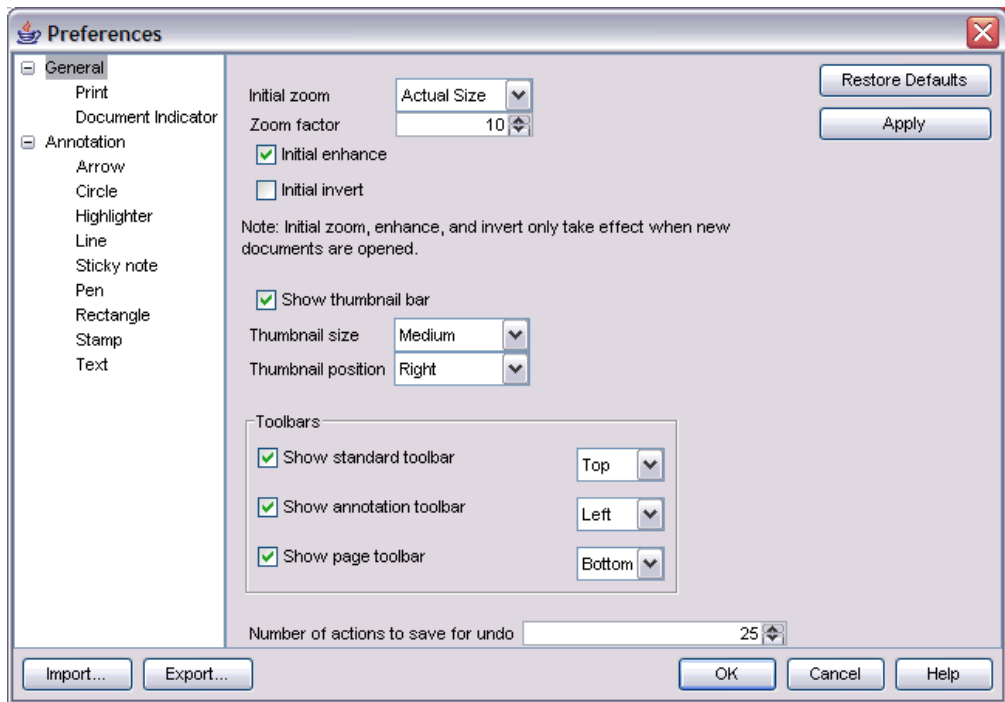


Figure 41. The preferences window

Following is a description of the list of buttons on the Preferences window:

#### Restore Defaults

Sets the preferences on the page to their default values.

#### Apply

Applies the changes for the preferences on the page.

#### Import

Imports preferences from a file.

**Export** Exports preferences to a file.

**OK** Applies changes for all preferences and close the dialog box.

#### Cancel

Closes the dialog box and restores all preferences that have not been applied to their values at the time the dialog box was opened.

You can change the configuration of pages and preferences on each page using the viewer configuration properties. The default properties include:

- `preference.elements=general,annotation`
- `preference.general=ps.document,ps.thumbnails,ps.toolbars,ps.general.common`
- `preference.general.elements=ps.printing,ps.appearance.doc.indicator`
- `preference.annotation=ps.annotation.common`
- `preference.annotation.elements=ps.arrow,ps.circle,ps.highlight,ps.line,ps.note,ps.pen,ps.rectangle,ps.stamp,ps.text`

Although the user only sees one set of preferences, there are actually two sets stored: System preferences and User preferences. The System preferences are

initially derived from the values in the viewer configuration properties. You can modify those by using the `importSystemPreferences` method, described in the following section. System preferences are the values used when the user clicks **Restore Defaults**. User preferences are created when a user changes system default values.

On Windows, system preferences are saved under `HKEY_LOCAL_MACHINE/JavaSoft/Prefs/com/ibm/mm/viewer` and user preferences are saved in the registry under `HKEY_CURRENT_USER/JavaSoft/Prefs/com/ibm/mm/viewer`.

For ease of importing and exporting, the preferences are written as an XML file. The schema used for this XML is described in the Java 1.5 preferences documentation. The key used is `com/ibm/mm/viewer`.

The preferences support is provided by the `CMBGenericDocViewer` class using the following methods:

- `importSystemPreferences(InputStream is)`
- `importUserPreferences(InputStream is)`
- `haveUserPreferencesChanged()`
- `exportSystemPreferences()`
- `exportUserPreferences()`

For more implementation details, for developing an application, see the `CMBGenericDocViewer` class information in the *Application Programming Reference* in the information center.

By default the **Preferences** tool is on the **Operation** toolbar, also referred to as the standard toolbar. If this toolbar is removed by not selecting the **Show standard toolbar** checkbox on the Preferences window, use the **Preferences** tool shortcut key, `ctrl+shift+P` to bring up the Preferences window.

## Print preferences

Clicking on **Print** in the tree view shows the general print preferences.



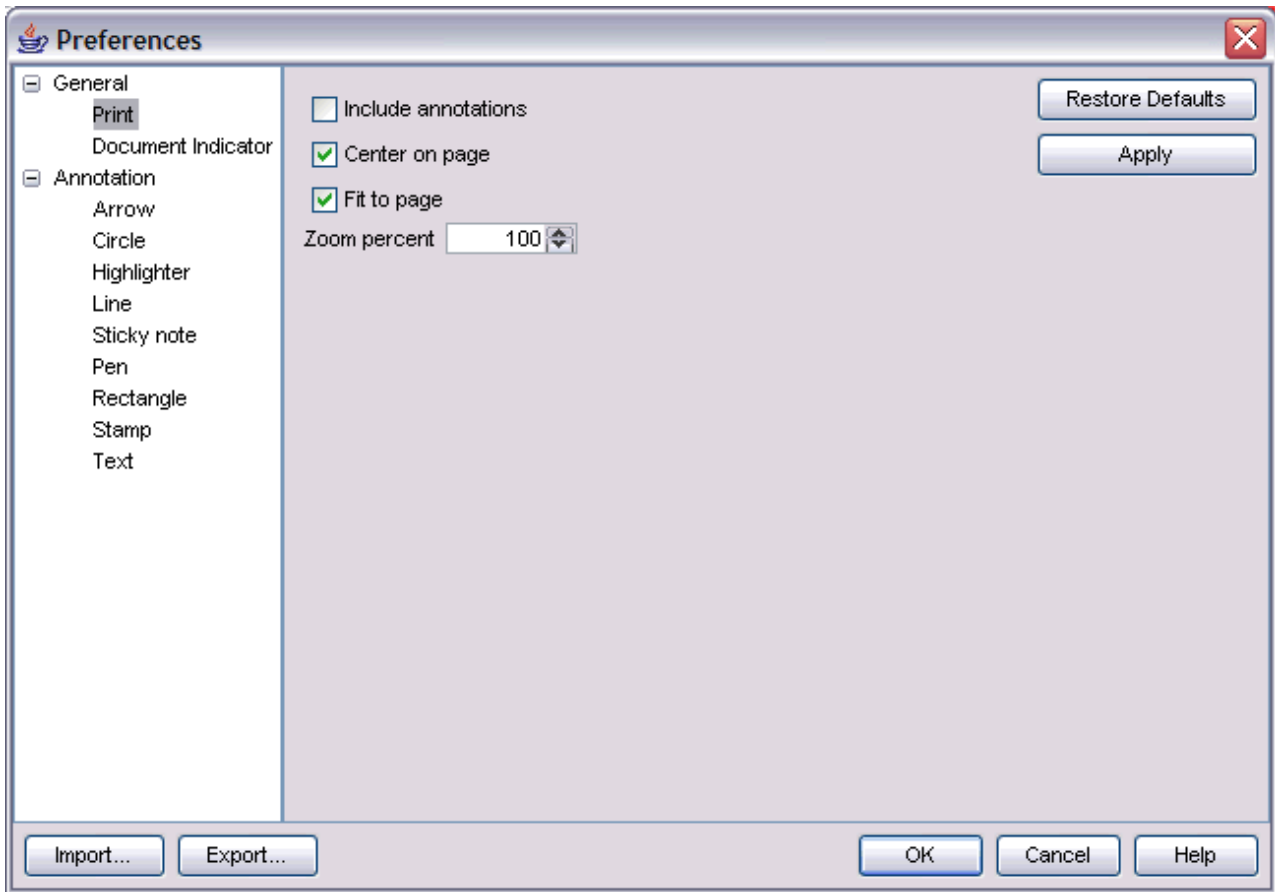


Figure 42. Printing preferences in Java viewer

You specify the settings on the Print dialog box using the new printing preferences. These include printing a document with or without annotations, and scaling and centering the document page. Use the Preferences window to make changes to the printing preferences.

The preference set, `ps.printing`, is included in the default viewer configuration properties file, `cmbviewerconfiguration.properties`. This preference set contains the printing preferences that are used as initial settings for the print dialog boxes. The set includes the following preferences:

**Printing.annotations.include**

Controls the check box to include annotations on the print dialog box.

**Printing.centerOnPage**

Controls the center on page check box on the Print dialog box.

**Printing.fitToPage**

Controls the fit to page check box on the Print dialog box.

**Printing.zoom**

Control the zoom percent on the Print dialog box. This is an integer from 25 to 500 that specifies a zoom percentage of document when printed, with a default of 100.

The classes, `CMBGenericDocPrintStatusEvent`, `CMBGenericDocPrintStatusEvent` and `CMBGenericDocPrintStatusListener` gather feedback on printing progress. See them in the *Application Programming Reference* in the information center.

## Printing

You can also print multiple documents in one step. Print All Documents viewer action button prints all documents that are currently open.

You specify the settings on the Print windows using the new printing preferences. These include printing a document with or without annotations, and scaling and centering the document page. Use the Preferences window to make changes to the printing preferences. You can fit each page of the document on a printed page or specify the percent to zoom. The fit to page option takes precedence. If fit to page is selected, the document pages are fit to the page when printed, otherwise the zoom percent value is applied.

In addition, because there will be many documents printed, printing multiple documents will queue the rendering of documents in order to manage memory usage.

New classes have been added for gathering feedback on printing progress. See the new classes `CMBGenericDocPrintStatusEvent` and `CMBGenericDocPrintStatusListener` in the *Application Programming Reference* in the information center.

## Document indicator preferences

Clicking on Document Indicator in the tree view provides the following:

- Ability to change the font style and color of the document name on document selection tabs within the viewer to indicate the documents that have already been viewed
- A timer mechanism to delay the change in font/color because a document might be only briefly viewed when navigating from one document to other
- Control over the font style, color, and time of delay in the viewer preferences

Figure 43. Document indicatorDocument indicator window

The preference set `ps.appearance.doc.indicator` includes the following preferences:

### **Appearance.tab.indicate**

Enables or disables the setting of the font style and color for documents that have been viewed. The default is true.

### **Appearance.tab.fontstyle**

Specifies the font style for the document tabs for documents that have been viewed. This preference is used if `Appearance.tab.indicate=true`. The default value is bold. The valid values are bold, italic, bold\_italic, plain.

### **Appearance.tab.color**

Specifies the color of the text for the document tabs for documents that have been viewed. This preference is used if `Appearance.tab.indicate=true`. The default value is green (`#00ff00`). The valid values are hex strings that represents rgb value as `#rrggbb`.

### **Appearance.tab.delay**

Specifies how many seconds a document needs to be viewed before the tab is indicated. This preference is used if `Appearance.tab.indicate=true`. The default value is 10. The valid value will be an integer from 0 to 1024.

Two new methods, `hasDocumentBeenViewed` and `setDocumentViewed` have been added to `CMBGenericDocViewer`. See the new methods in `CMBGenericDocViewer` in the *Application Programming Reference* in the information center for more information.

## Annotation preferences

Clicking on **Annotation** in the tree view shows the general annotation preferences:

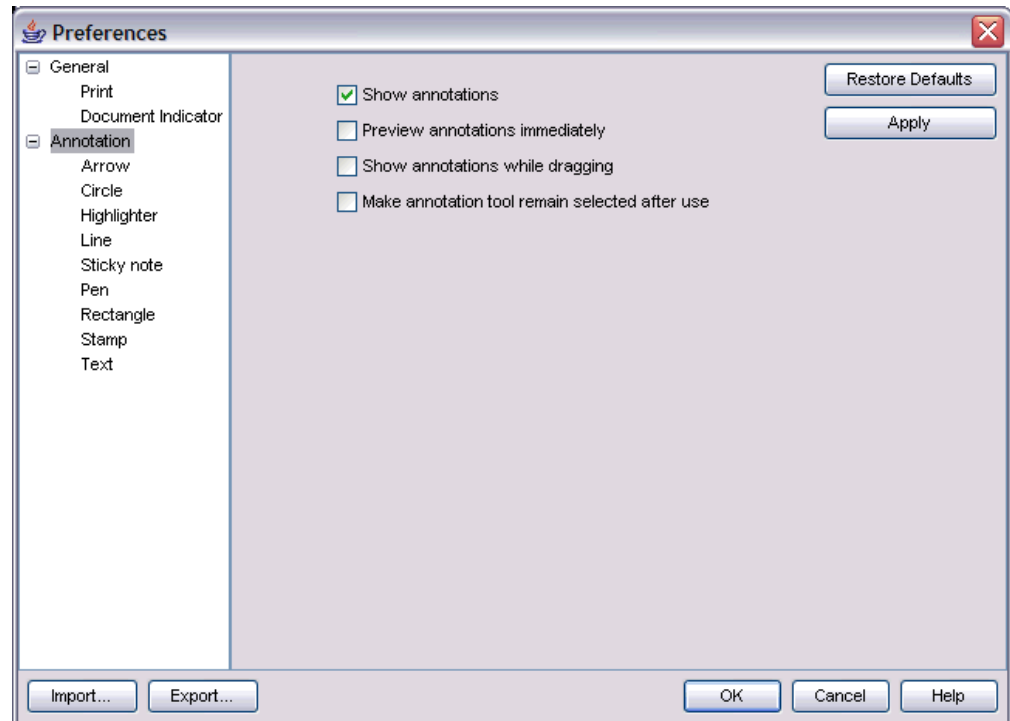


Figure 44. The list of general annotation preferences

Clicking on an individual annotation type displays the preferences for that annotation type. For example, see the Sticky Note window in Figure 45 on page 471:

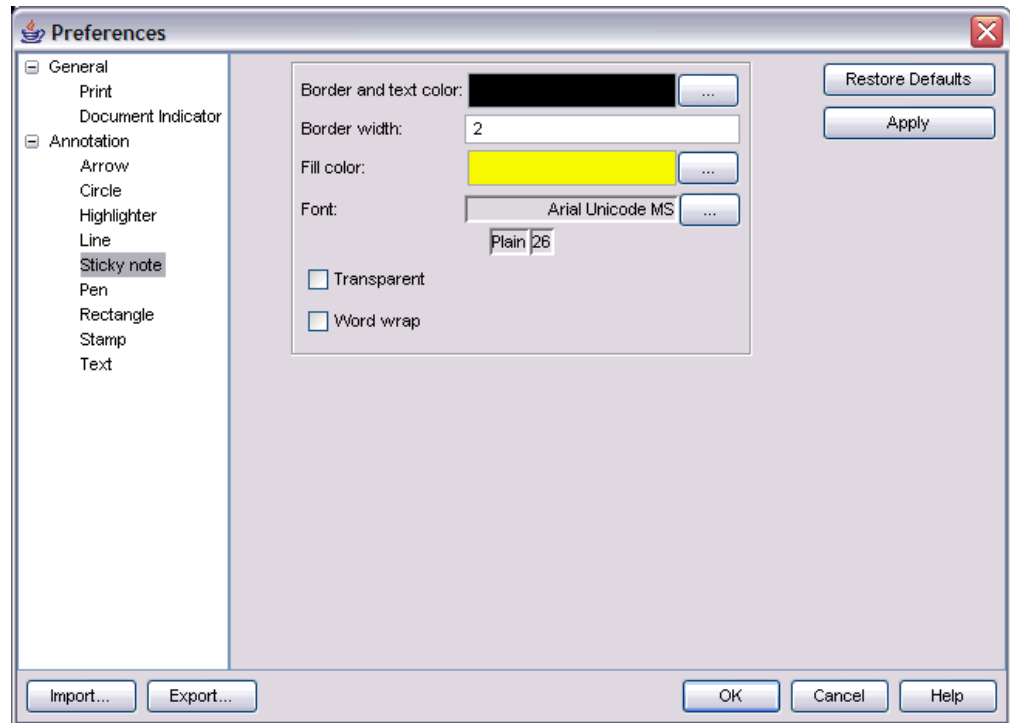


Figure 45. Preferences for the sticky note annotation

## Java viewer layout tools

To enable displaying multiple views of one or more documents simultaneously in the Java viewer applet, the Java viewer toolkit provides the following layout tools:

### Tabbed

Displays one document in the viewer, and provides tabs to select other documents. This is the default view. Shortcut key: Ctrl – 6.

### Split Horizontal

Divides the viewer into left and right viewer areas. You can view different documents within each area using the drop-down lists. The side-by-side layouts are displaying two views onto the same set of documents in the viewer. Annotations or page manipulations performed in one pane will also appear on the same document in the other pane. Shortcut key: Ctrl – 7.

### Split Vertical

Divides the viewer into a top and a bottom viewing area. Shortcut key: Ctrl – 8.

### Thumbnails Only

Displays the thumbnails for a document. You can select documents in a tabbed pane, and jump to the pages by double-clicking them. Shortcut key: Ctrl – 9.

The Java viewer toolkit also enables side-by-side viewing. To enable side-by-side viewing, see the methods that are added to `CMBGenericDocViewer`, `CMBAnnotationServices`, `CMBAnnotationView`, `CMBPageAnnotation`, `CMBAnnotationSet`, and `CMBAnnotationSelectedEvent` in the *Application Programming Reference* in the information center.

To define toolbars and menus, the layout tools that are included in the viewer configuration properties are:

**layout\_tabbed**

Action to switch to tabbed layout.

**layout\_split\_horizontal**

Action to switch to side-by-side horizontal layout.

**layout\_split\_vertical**

Action to switch to side-by-side vertical layout.

**layout\_thumbnails\_only**

Action to switch to thumbnails only layout.

---

## Java document viewer limitations

There are some limitations of the Java document viewer.

Following is a list of known limitations in the Java document viewer:

**The Intel® Graphic Controller shortcut key conflicts with viewer shortcut**

When a client that is using the Java viewer toolkit, such as the eClient applet viewer, and the Intel Graphic Controller application are active, there is a conflict with the Ctrl+Alt+I shortcut key. This shortcut is used in the viewer to invert a document.

The Graphic Controller has hot keys enabled by default. When the hot keys feature is disabled in the Intel Graphic Controller, the viewer keys work fine.

To modify the eClient applet viewer to use different keys, complete the following steps:

1. Add a `Circle.key=control alt I` entry to the eClient `cmbViewerConfigurations.properties` file.
2. Set the key to another choice, such as `Circle.key=shift 0`.

**JAWS screen reader Version 8.0 interferes with Java viewer shortcuts that begin with the Alt key**

The JAWS screen reader interferes with the shortcuts in the Java viewer that begin with the Alt key. In the Java viewer, Alt+Enter launches annotation properties, Alt+F4 closes any window, Alt+Left Arrow or Right Arrow moves the annotations. Trying to move annotations using Alt+Up Arrow or Down Arrow can cause the annotations to jump or not move.

To move the annotations, close the JAWS screen reader and move the annotations using the keyboard keys. If you have JAWS open, move the annotations using the mouse.

**Java viewer and eClient applet viewer Print window does not have focus by default**

By default, when the Print window opens in the viewer, it does not have focus. Therefore, you cannot navigate the window using the keyboard (Tab to navigate or Alt + mnemonic character to select a particular field). To work around this problem, click the window to give it focus. Press Alt+Tab to switch to a different window.

---

## Working with the JSP tag library and controller servlet

DB2 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 DB2 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 DB2 Information Integrator for Content.

---

### Using the tag library

This example shows a JSP sample that is using the search templates tag to get a list of search templates.

#### Example

```
<%@ 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>Available Search Templates</td>
 </tr>
 <tr>
 <td>Search Templates</td>
 </tr>
 <tr>
 <td><%= searchtemplate.getName() %></td>
 </tr>
 </table>
 </body>
</html>
```

The taglib directive declares that the page uses the DB2 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 48 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 48. 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

This section summarizes the tag library.

### Connection related tags

This sections lists all the 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**

This section lists all the 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

This section lists the 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

This section lists the item related tags.

**<cmb:itemattributes item="item"> attrname ... attrtype... attrvalue... </cmb:itemattributes>**

This tag iterates through the attributes of an item.

**item** Specify the name of a variable of type CMBItem that contains the item.

**attrname**

A string variable to contain the name of the attribute.

**attrtype**

A string variable to contain the attribute type.

**attrvalue**

A string variable to contain the value of the attribute.

**<cmb:itemcontents " data="data" item="item"> content... </cmb:itemcontents>**

This tag iterates through the contents of an item.

**data** Specify the name of a variable of type CMBDataManagement.

**item** Specify the name of a variable of type CMBItem that contains the item.

**content**

A variable of type CMBObject for the item's contents.

**<cmb:itemnotelogs " data="data" item="item">notelog ... </cmb:itemnotelogs>**

This tag iterates through the note logs of an item.

**data** Specify the name of a variable of type CMBDataManagement.

**item** Specify the name of a variable of type CMBItem that contains the item.

**notelog**

A variable of type CMBObject to contain the item's note log.

**<cmb:itemprivileges data="data" item="item">privilege ... </cmb:itemprivileges>**

This tag iterates through the privileges of an item.

**data** Specify the name of a variable of type CMBDataManagement.

**item** Specify the name of a variable of type CMBItem that contains the item.

**privilege**

A variable of type CMBPrivilege to contain the item's privilege.

**<cmb:itemresources data="datamanagement" item="item"> resource...</cmb:itemresources>**

This tag iterates through the resources of an item.

**data** Specify the name of a variable of type CMBDataManagement.

**item** Specify the name of a variable of type CMBItem that contains the item.

**resource**

A variable of type CMBResources to contain the item's resource.

**<cmb:unmappeditem data="data" item="item">unmappeditem...</cmb:unmappeditem>**

This tag returns an unmapped item from the given mapped item.

**data** Specify the name of a variable of type CMBDataManagement.

**item** Specify the name of a variable of type CMBItem that contains the mapped item.

**unmappeditem**

A variable of type CMBItem to contain the unmapped item.

**<cmb:viewdata data="data" item="item">viewdata... </cmb:viewdata>**

This tag returns a view of an item.

**data** Specify the name of a variable of type CMBDataManagement.

**item** Specify the name of a variable of type CMBItem that contains the item.

**viewdata**

A variable of type CMBViewData to contain the viewable data.

## Folder related tags

This section lists the 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

This section lists the document related tags.

**<cmb:viewerddocuments docservices="docservices">document ...  
</cmb:viewerddocuments>**

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.

---

## DB2 Information Integrator for Content controller servlet

DB2 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

This section highlights some of the aspects of the servlet that you can use.

Some of the aspects of the servlet that you can use are:

### Connection pooling

The controller servlet uses DB2 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 DB2 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 `cmbervletjsp.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
```

**Important:** The feature to add custom actions to precede or follow any predefined action is not supported.

The naming convention used for all predefined actions is `com.ibm.mm.servlets.CMBActionAction`, 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 DB2 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                                                                                                                                                                                                                                                                                          |
|-----------------------------|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>servletPropertiesURL</b> | URL                       | The location of the cmbervlet.properties file                                                                                                                                                                                                                                                        |
| <b>defaultServerType</b>    | Fed, ICM, OD, V4, IP, ... | Default logon information. This, along with <b>defaultServer</b> , <b>defaultUserid</b> , and <b>defaultPassword</b> 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.                          |
| <b>defaultServer</b>        |                           | Default logon information.                                                                                                                                                                                                                                                                           |
| <b>defaultUserid</b>        |                           | Default logon information.                                                                                                                                                                                                                                                                           |
| <b>defaultPassword</b>      |                           | Default logon information.                                                                                                                                                                                                                                                                           |
| <b>connectionpool</b>       | Boolean: TRUE   FALSE     | To enable connection pooling                                                                                                                                                                                                                                                                         |
| <b>maxfreeconnection</b>    | integer                   | Maximum number of connections available in a connection pool.                                                                                                                                                                                                                                        |
| <b>minfreeconnection</b>    | integer                   | Minimum number of connection available in a connection pool                                                                                                                                                                                                                                          |
| <b>timeout</b>              | integer                   | The time duration (in milliseconds) after which a free connection will be disconnected and destroyed.                                                                                                                                                                                                |
| <b>noSessionPage</b>        | 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 DB2 Information Integrator for Content to work even if the user must log on. |
| <b>timedOutPage</b>         | URL                       | This is a page to display if the session has timed out due to inactivity.                                                                                                                                                                                                                            |
| <b>serverErrorPage</b>      | URL                       | This is a page to display if an error has occurred in accessing a server.                                                                                                                                                                                                                            |
| <b>connectFailedPage</b>    | 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.                                                                                                            |

| Application parameter        | Values                 | Description                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>tracelevel</b>            | 0, 1, or 2             | To indicate the level of tracing, as follows: <ul style="list-style-type: none"> <li>• 0 - log nothing</li> <li>• 1 - log exceptions (default)</li> <li>• 2 -log exceptions, alert messages, WebSphere Application Server headers and attributes, DB2 Information Integrator for Content ini files, JVM system properties, DB2 Information Integrator for Content internal trace information</li> </ul> |
| <b>connectiontype</b>        | 0, 1, or 2             | The location of the DB2 Information Integrator for Content database and content server runtimes: <ul style="list-style-type: none"> <li>• 0 - local (default)</li> <li>• 1 - remote</li> <li>• 2 -dynamic</li> </ul>                                                                                                                                                                                    |
| <b>cmbclient</b>             | URL                    | Location of cmbclient.ini                                                                                                                                                                                                                                                                                                                                                                               |
| <b>cmbcs</b>                 | URL                    | Location of cmbcs.ini                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>serviceconnectiontype</b> | 0, 1, or 2             | Location of services runtimes <ul style="list-style-type: none"> <li>• 0 - local (default)</li> <li>• 1 - remote</li> <li>• 2 -dynamic</li> </ul>                                                                                                                                                                                                                                                       |
| <b>cmbsvclient</b>           | URL                    | Location of cmbsvclient.ini                                                                                                                                                                                                                                                                                                                                                                             |
| <b>cmbsvcs</b>               | URL                    | Location of cmbsvcs.ini                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>cmbcc2mime</b>            | URL                    | Location of cmbcc2mime.ini                                                                                                                                                                                                                                                                                                                                                                              |
| <b>cachedir</b>              | name of a directory    | Directory to cache documents during document conversion                                                                                                                                                                                                                                                                                                                                                 |
| <b>jnitrace</b>              | 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)                                                                                                                                                                                                                                                                            |
| <b>conversion</b>            | Boolean: TRUE or FALSE | If TRUE, documents are converted to formats that can be displayed in a browser on middle tier, if possible. If FALSE, the original document, unconverted, is sent to the browser.                                                                                                                                                                                                                       |
| <b>maxresults</b>            | integer                | Maximum hits returned; -1 (default) means all hits.                                                                                                                                                                                                                                                                                                                                                     |
| <b>valuedelimiter</b>        | character              | Defines the character that will delimit values in search criteria. The default is locale dependent and is comma (,) for US English.                                                                                                                                                                                                                                                                     |

| Application parameter              | Values                    | Description                                                                                                                                                                                                                                                                                                           |
|------------------------------------|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>conversion.&lt;mimetype&gt;</b> | <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. |
| <b>nameseparator</b>               | 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.

## Example

```
// check to see if connection pooling is enabled
String name = "connectionpool";
Properties props = (Properties) application.getAttribute(
 "cmbServletProperties");
String value = props.getProperty(name);
// "true" if enabled, "false" otherwise
```

## Request parameters

The servlet understands the following request parameters. Additional parameters can be specified, for use in the reply JSP.

### General

#### **action = action**

The action to be performed. Additional parameters allowed are based on the action and are described below.

This parameter is optional. If it is not specified, the reply page will be executed by the servlet, after performing standard setup, such as checking the connection for time-out and logon.

#### **reply=<URL>**

Optional. Forwards to the JSP specified in this parameter rather than the JSP defined as the reply for the action in the `cmbServlet.properties` file. If the action parameter is not specified, and reply is specified, the reply page is executed by the controller servlet after performing standard setup, such as checking the connection for timeout and logon.

## Connection Related

**action=logon** **serverType=<>** **server=<>** **userid=<>** **password=<>**  
**[connstring=<>] [configstring=<>]**

Login to a server. You must specify the server type, server name, user ID, and password. The connect string and init string are optional and are different depending on the type of server.

**action=logoff** **[endSession=<true|false>]**

Logout of the server. The session is also ended by default.

## Search related

**action=searchTemplate** **template=<>** **{<criteriaName>.op=<>**  
**<criteriaName>=<>}**

Perform a search using the specified search template and criteria values.

**action=searchEntityentity=<>** **{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 re-indexed. 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

This section lists the servlet toolkit function matrix.

*Table 49. Servlet function matrix*

| Action         | CMv8 | VI/400 | IP/390 | OD/Wkstn | OD/390 |
|----------------|------|--------|--------|----------|--------|
| logon          | Y    | Y      | Y      | Y        | Y      |
| logoff         | Y    | Y      | Y      | Y        | Y      |
| searchTemplate | N/A  | N/A    | N/A    | Y        | Y      |
| searchEntity   | Y    | Y      | Y      | Y        | Y      |
| searchQuery    | Y    | Y      | Y      | Y        | Y      |
| lock           | Y    | Y      | Y      | N/A      | N/A    |
| unlock         | Y    | Y      | Y      | N/A      | N/A    |
| createItem     | Y    | N/A    | N/A    | N/A      | N/A    |
| retrieveItem   | Y    | Y      | Y      | Y        | Y      |
| updateItem     | Y    | Y      | Y      | N/A      | N/A    |

Table 49. Servlet function matrix (continued)

| Action               | CMv8 | VI/400 | IP/390 | OD/Wkstn | OD/390 |
|----------------------|------|--------|--------|----------|--------|
| deleteItem           | Y    | Y      | Y      | N/A      | N/A    |
| addContent           | Y    | N/A    | N/A    | N/A      | N/A    |
| getContent           | Y    | Y      | Y      | Y        | Y      |
| updateContent        | Y    | Y      | Y      | N/A      | N/A    |
| deleteContent        | Y    | Y      | Y      | N/A      | N/A    |
| addNoteLog           | Y    | Y      | N/A    | N/A      | N/A    |
| updateNoteLog        | Y    | Y      | N/A    | N/A      | N/A    |
| deleteNoteLog        | Y    | Y      | N/A    | N/A      | N/A    |
| addItemToFolder      | Y    | Y      | N/A    | N/A      | N/A    |
| removeItemFromFolder | Y    | Y      | N/A    | N/A      | N/A    |
| viewDocument         | Y    | Y      | Y      | Y        | Y      |
| viewPage             | Y    | Y      | Y      | Y        | Y      |

**Notes:**

**Y** Function is supported

**N/A** Function is not supported

The logon, logoff, getContent, retrieveItem, viewDocument and viewPage actions are supported on all the content servers.

Table 50. Servlet function matrix Continued

| Action               | Fed |
|----------------------|-----|
| logon                | Y   |
| logoff               | Y   |
| searchTemplate       | Y   |
| searchEntity         | Y   |
| searchQuery          | Y   |
| lock                 | Y   |
| unlock               | Y   |
| createItem           | N/A |
| retrieveItem         | Y   |
| updateItem           | Y   |
| deleteItem           | Y   |
| addContent           | Y   |
| getContent           | Y   |
| updateContent        | Y   |
| deleteContent        | Y   |
| addNoteLog           | Y   |
| updateNoteLog        | Y   |
| deleteNoteLog        | Y   |
| addItemToFolder      | N/A |
| removeItemFromFolder | N/A |

Table 50. Servlet function matrix Continued (continued)

| Action       | Fed |
|--------------|-----|
| viewDocument | Y   |
| viewPage     | Y   |

**Notes:**

**Y** Function is supported

**N/A** Function is not supported

The logon, logoff, getContent, retrieveitem, viewDocument and viewPage actions are supported on all the content servers.

---

## Troubleshooting content management applications

This section contains the following troubleshooting topics.

---

### Java, C++, and DB2 Information Integrator for Content connector logging

You can use the log control utility in the system administration client to configure the log settings for the Java and C++ APIs.

The connector logging utilities store all program exceptions, including exceptions that are not errors. Occasionally, error messages appear in the log file that are not sent to the user. In some cases, the API or user application can recover or continue without user interaction required. The `${user.name}.dklog.log` log file contains the log results from DB2 Information Integrator for Content connector actions.

The `${user.name}` is the machine system logon user ID that is used to qualify the log output. The system logon user Id will be used to replace `${user.name}` which will help to avoid output collisions from multiple users.

**Important:** When reading the log files, note the context within which the exceptions and messages are logged. Also, when debugging, concentrate the log results on current errors and events before them by defining constraints in the API log file. More limited searches can improve system performance.

If the system administration client is not available, you can configure the APIs by changing values in the `cmblogconfig.properties` file. In the log control utility, you can change the Java and C++ default log file names, log file path settings, maximum file size, and the level of error information to send to the log file. For assistance using the log control utility to configure API logging, use the system administration online help.

### Java connector logging

Java users can use two log managers: the default and the `log4j-1.2.8.jar`. The product installation sets the log manager to `log4j-1.2.8`. You can configure and use only one log manager at a time.

The same configuration file `cmblogconfig.properties` is used to control the type of log manager used and the configuration specific to each type of log manager. For more information about the log manager, see the section for the log manager that you want to use in `cmblogconfig.properties` file.

When the connector logging utility is first instantiated, it searches the CLASSPATH of the Java virtual workstation instance to find the `ibmcmconfig.properties` file. The path to the working directory is listed in *IBMCMWorkingDirectory*. The log configuration file `cmblogconfig.properties` can be found from the `cmgmt` subdirectory from the working directory.

## C++ connector logging

C++ has one log manager. C++ references the same log configuration file as Java. However, the DB2 Information Integrator for Content C++ connectors reference only the default log manager logging settings.

The C++ connector depends on the environment variable *IBMCMROOT*.

The *ibmcmconfig.properties* must be located in the *IBMCMROOT/cmgmt* directory.

## Modifying the logging configuration file for the DB2 Information Integrator for Content connectors

The configuration file is installed in the *cmgmt* directory of the DB2 Content Manager working directory. The file is named *cmblogconfig.properties*.

The *cmblogconfig.properties* file contains the following default settings. Do not to change these default settings in case the configuration file cannot be found or other errors occur with user-defined settings:

- *Log4j* is set as the default log manager. The default log file name is *userid.dklog.log*. The default location of the log file is *IBMCMWorkingDirectory/log/connectors*.
- The *dklog.log* file is placed in the current working directory where an DB2 Information Integrator for Content-enabled application is run.
- The logging priority is set to *Error*.
- The maximum number of exceptions of the same error message ID to allow is five (5).

To update the settings in the *cmblogconfig.properties* file:

1. Open the *cmblogconfig.properties* file in a text editor.
2. Change the settings as required to meet your system logging requirements.

| Option                                  | Description                                                          |
|-----------------------------------------|----------------------------------------------------------------------|
| Section 0 - Global Settings             | Determines maximum exception count.                                  |
| Section 1 - Log Manager Factory Setting | Determines whether you use the default log manager or <i>log4j</i> . |

| Option                                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Section 2 - Default Log Manager Setup | <p>Section 2 has three subsections:</p> <ul style="list-style-type: none"> <li>• Section 2.1 - Specify Log Priority. Eight priority settings are available. The default priority setting is Error.</li> <li>• Section 2.2 - Log Output Destination Setting. Three settings are available: <ol style="list-style-type: none"> <li>1. Log to a file</li> <li>2. Log to Standard Error</li> <li>3. Log to Standard Console</li> </ol> <p>The default setting is Log to file.</p> </li> <li>• Section 2.3 - Log File Name Setting. Use only when the option in section 2.2 is set to Log to file. <p>The default log file name is <i>userid.dklog.log</i>.</p> <p>The default location of the log file is <i>IBMCMWorkingDirectory/log/connectors</i>. The log file path must include double backslash (\\) or single slash (/) characters in place of single backslashes if any are used. If you do not specify a path, the log is always located in the current working directory where a problem is executed.</p> <p>The log file location is controlled by the following settings:</p> <ul style="list-style-type: none"> <li>– DKLogOutputFileName for default logger (both C++ and Java)</li> <li>– Log4j.appender.apiAppender.File for log4j</li> </ul> <p>The maximum number of backup files to keep by setting DKLogOutputFileSave for default logger, both C++ and Java or log4j.appender.apiAppender.MaxBackupIndex for log4j.</p> <p>The maximum file size by setting DKLogOutputFileSize for default logger, both C++ and Java or log4j.appender.apiAppender.MaxFileSize for log4j.</p> </li> </ul> |

### 3. Optional: Modify the following priority levels in cmblogconfig.properties file.

| Option           | Description                                                                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DISABLE          | Disables logging.                                                                                                                                                   |
| FATAL            | Notification that the program encountered unrecoverable errors and must cease operating immediately. Stopping the program is not initiated by the logging facility. |
| ERROR            | Notification that the program encountered recoverable or unrecoverable errors, but is able to continue operating.                                                   |
| PERF             | Collects output information for measuring performance.                                                                                                              |
| INFO             | Provides significant event messages, such as successful logon.                                                                                                      |
| TRACE_NATIVE_API | Logs parameters and return data information before and after a native call.                                                                                         |

| Option           | Description                                                                                                                                      |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| TRACE_ENTRY_EXIT | Signals entries and exits of program modules (or code blocks).                                                                                   |
| TRACE            | Records additional diagnostic information, such as program state changes, function parameter information, and function return value information. |
| DEBUG            | Displays information for debugging errors.                                                                                                       |

4. Save the file.

---

## Receiving an error when compiling C++ applications that are Unicode enabled

### Possible cause

The Content Manager V8 C++ APIs do not support Unicode.

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).

### 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.

The CM product installs using English as the default language. However, after the installation is complete, to display an item's attribute descriptions in the local language on a client application, 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

### Symptoms

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'.

## Cause

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 return code 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.

---

## Receiving an error when WebSphere Application Server connection pooling is enabled

### Symptom

When running an application that uses the container transaction type for the session bean and WebSphere Application Server connection pooling is enabled, you receive the following error:

```
[11/23/05 18:50:52:934 CST] 6be0947b SystemErr???? R???????????????
java.sql.SQLException: DSR9350E: Operation Connection.commit
is not allowed during a global transaction.
```

### Possible Cause

The DB2 Content Manager API does not support Java beans when WebSphere Application Server connection pooling is enabled.

### Action

To avoid this error, you can use one of the following solutions:

- Use the TX\_NOT\_SUPPORTED property with a container-managed session bean.
- Do not use Java beans with WebSphere Application Server connection pooling when using the DB2 Content Manager APIs.

---

## Receiving a search error in the Thai language when database setup is incorrect

### Symptoms

When you do a text search of Thai language content, you can perform one word searches or phrased searches. When you perform a phrased search, you get an error.



## Solution

To support phrased searches in Thai content in DB2 Content Manager, the database and the text index must be created in one of the following ways:

- Create the DB2 database in the Thai codepage or in Unicode.
- Create the text index in the Thai code page with the language specified as TH\_TH or in the Thai code page CCSID 1208.

To search for one Thai word by using the advanced text search syntax:  
`/MyItemTextView[contains-text(ICMPARTS/@TIEREF, " 'Thai word' ")=1]`

To search for Thai phrases by using the DB2 Thai-specific advanced text search syntax: `/MyItemTextView[contains-text-db2(ICMPARTS/@TIEREF, \" IS ABOUT TH_TH 'Thai phrase' \")=1]`

---

## Receiving an error when using ICM text search for Thai phrases in DB2

### Symptoms

When you use the IBM® DB2® Content Manager Windows client, or the DB2® Content Manager APIs to text search for Thai language phrases, you do not always get the expected results back when the database backend is DB2.

### Cause

The default query string that is generated by the DB2® Content Manager Windows client for text search is in the following format: `/MyItemTextView[contains-text(ICMPARTS/@TIEREF, " 'Thai phrase' ")=1]`

This is the basic search syntax and only supports searching on only one Thai word at a time.

### Action

Modify your query string to take advantage of the advanced DB2 text search syntax for searching Thai phrases (more than one word with no spaces in between). `"/MyItemTextView[contains-text-db2(ICMPARTS/@TIEREF, \" IS ABOUT TH_TH 'Thai phrase' \")=1]"`

If you are using the DB2 Content Manager Windows client, use the advanced text search option to specify 'IS ABOUT TH\_TH' in the optional parameters box before your search. For more information about the IS ABOUT parameter, see the *IBM DB2 Net Search Extender Administration and User's Guide*.

---

## Cannot add, store, retrieve, or update a resource item

### Symptoms

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.

## Causes

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.

Paste the sample URL into a Web browser, `http://smith.stl.ibm.com:80/icrm/ICMResourceManager`. 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 (:).

```
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 causes

The DB2 Net Search Extender (DB2 UDB Version 8) password is either not set, or set incorrectly.

## Action

Reset the password in the DB2 Content Manager system administration client.

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 displaying the right pane.
4. Click the **Features** tab.
5. Enter the correct DB2 Net Search Extender user ID and password.

---

## Cannot download ICMBASESTREAM part type content

### Symptom

If you create a document item type and add an ICMBASESTREAM part type to that document, you must have already defined a DB2 Content Manager VideoCharger collection on the server or you cannot download the content. If you attempt to download the content, you will get the following error: R com.ibm.mm.beans.CMBException: DGL3980A: The resource manager returned an invalid response

### Action

An ICMBASESTREAM part type must always be associated with a VideoCharger collection. When you are associating parts that are not VideoCharger collections, use the ICMBASE part type.

---

## Text search does not return results for documents created in certain languages

### Symptom

Text search does not return results for documents created in certain languages.

### Action

You must enable text search on non-European or white-space delimited languages, by specifying the LEXER preference or using CTX\_SYS.BASIC\_LEXER.

Create a LEXER preference that is appropriate for the language of the documents in the index.

To index Japanese documents, you must create a LEXER preference by completing the following steps:

1. Connect to the database as a DB2 Content Manager administrative user.  
Example using sqlplus: `sqlplus userid/password@dbname`
2. Create the preference: `exec ctx_ddl.create_preference('MY_JAPANESE_LEXER','japanese_lexer');`
3. The preference you created will display in the list in the system administration client. You need to exit the system administration client then restart the system administration client again. Under LEXER, select the preference you created when you create the index.

---

## DKString DKLobICM::getContent () const not recommended for ICM connector

### Possible cause

The `DKString DKLobICM::getContent()` const method is not recommended for the ICM connector, because if the instance of `DKString` contains binary data, and you call string functions like `substring()` on the instance, you might receive incorrect results. You should not use binary data with a `DKString` class.

### Action

Use the `DKByteArray* DKLobICM::getContent()` method.

---

## Cannot load user exit

### Symptom

I cannot load my user exit.

### Possible cause

You might not have set the `$ICMDLL` variable.

### Action

The user exit must be located in the directory `$ICMDLL/ICMNLSDDB/DLL`, where `ICMNLSDDB` is the name of the library server.

You must also complete the preinstallation steps to set the `$ICMDLL` environment variable on the DB2 Content Manager instance. You can find the steps in the *Planning and Installing Your Content Management System* information.

To point `$ICMDLL` to the correct location:

1. Connect to the library server database.
2. Select `pathicmdl` from `icmstsyscontrol`.
3. Update `db2InstHome/sql/lib/userprofile`:  
`ICMDLL= <value from #2 above>`  
`export ICMDLL`
4. Run `recycle db2` instance.
5. Place the following library server user exits in the location you point to in the previous steps:

- ACL General Privilege User Exit: icmgenxt
- Full-Text Search User Exit: icmdecxt

---

## Cannot call some administration APIs in an explicit transaction

### Symptom

If you call the specific APIs, you will receive the following exception:  
DKUsageError DGL0594A: This operation cannot be performed within an explicit transaction.

### Possible Causes

You cannot call some administration APIs in an explicit transaction because of the restrictions from the underlying database. In some cases, calling the APIs can create or delete tables, which causes locks that produce deadlocks or timeouts.

The following examples of code might cause the exception to be thrown when you call the del() method:

```
//..Get an iterator to the list of item types in the system
dkCollection itemTypeDefs = dsDefICM.listEntities();
dkIterator iter = itemTypeDefs.createIterator();
//..Pick the first item type
DKItemTypeDefICM itemType = (DKItemTypeDefICM) iter.next();

//..Start an explicit transaction
_ds.startTransaction();

//..Attempt to delete an item type in an explicit transaction
// This will throw a DKUserError
itemType.del();

//..Commit the explicit transaction
_ds.commit();
```

Following is a list of methods that are not allowed in explicit transactions:

```
DKDatastoreDefICM.add(dkEntityDef itemTypeObj); //create item type or view
DKDatastoreDefICM.del(dkEntityDef itemTypeObj); //delete item type or view
DKDatastoreDefICM.update(dkEntityDef itemTypeObj); //update item type or view
DKDatastoreDefICM.add(DKComponentTypeIndexDefICM componentTypeIndexObj);
//create component type index
DKDatastoreDefICM.del(DKComponentTypeIndexDefICM componentTypeIndexObj);
//delete component type index
DKDatastoreDefICM.update(DKComponentTypeIndexDefICM componentTypeIndexObj);
//update component type index
DKDatastoreDefICM.rebuildComponentType(String compTypeName);
//rebuild component type

DKComponentTypeIndexDefICM.add();
DKComponentTypeIndexDefICM.del();

//..The restrictions apply to the following methods in
//DKComponentTypeDefICM and all of its subclasses,
//including DKItemTypeICM, DKComponentTypeViewDefICM,
//and DKItemTypeViewDefICM.
DKComponentTypeDefICM.add();
DKComponentTypeDefICM.update();
DKComponentTypeDefICM.del();

DKUserMgmtICM.add(dkUserDef userDef); //create an user
DKUserMgmtICM.del(dkUserDef userDef); //delete an user
DKUserMgmtICM.update(dkUserDef userDef); //update an user
```

**Important:** The item type restriction applies to DB2 Universal Database Content Manager systems.

---

## **Connect() API does not check for null or empty password in Trusted Logon mode**

### **Symptom**

DB2 Content Manager client applications that use the connect() API can authenticate user IDs without a password whenever Trusted Logon mode is enabled.

### **Cause**

When the DB2 Content Manager server is set up for Trusted Logon mode, any client application that uses the normal connect() method can log into the library server without providing a password because it assumes that the WebSphere Application Server performs the authentication. The connect() API does not check for a null or empty password. After the API passes the database authentication layer (as in, using the shared database connection ID in the INI file), the API calls the logon without a password. Therefore, when Trusted Logon mode is enabled, DB2 Content Manager allows the logon to succeed.

### **Action**

Always use the connectWithCredential() method in conjunction with Trusted Logon to force authentication through the WebSphere Application Server. The connectWithCredential() API validates the LTPA tokens before sending the user ID to the library server logon stored procedure.

---

## **Wildcard character search of the library server in z/OS can sometimes return incorrect data**

### **Symptom**

A wildcard character search of the library server in z/OS can sometimes return incorrect data. The error result from a search is similar to the following exception:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X !!! Exception !!! X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Name: com.ibm.mm.sdk.common.DKUsageError
Message: DGL0303A: Invalid parameter: Component Type View Identifier
= -2147483648
com.ibm.mm.sdk.common.DKUsageError: DGL0303A: Invalid parameter: Component
Type View Identifier = -2147483648
 at com.ibm.mm.sdk.common.DKDatastoreDefICM.retrieveComponentTypeView
(DKDatastoreDefICM.java:776)
 at com.ibm.mm.sdk.server.DKResultSetCursorICM.getDDOFromCursor
(DKResultSetCursorICM.java:1761)
 at com.ibm.mm.sdk.server.DKResultSetCursorICM.loadDDOBlock
(DKResultSetCursorICM.java:1942)
 at com.ibm.mm.sdk.server.DKResultSetCursorICM.setToNext
(DKResultSetCursorICM.java:466)
 at com.ibm.mm.sdk.server.DKResultSetCursorICM.fetchNext
(DKResultSetCursorICM.java:546)
 at com.ibm.mm.sdk.server.DKDatastoreICM.evaluate
(DKDatastoreICM.java:4320)
```

## Action

To fix this problem, apply the DB2 Universal Database z/OS PTF that corresponds to the level of DB2 Universal Database that you are using:

- PTF UK08338 (DB2 UDB zOS Release 710)
- PTF UK08339 (DB2 UDB zOS Release 810)

---

## Receiving a DKException message when query times out

### Symptom

Your DB2 Content Manager application is sending a query timeout value as one of the query options with all the DB2 Content Manager queries. When the query times out, you get a DKException message

### Solution

When a DB2 Content Manager query times out, the DKDatastoreICM query method will throw a DKQueryException with the error ID of DKMessageICM.DK\_ICM\_MSG\_QUERY\_TIMEOUT (7254).

### Example

See the following Java example to know the query exception in a timeout call.

```
dkResultSetCursor cursor = dsICM.execute(
 "/NOINDEX", DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
catch (DKException exc) {
 // Check for query timeout exception
 if (exc.getErrorId() == DKMessageICM.DK_ICM_MSG_QUERY_TIMEOUT){
 // Insert exception handling for timeout here...
 } else {
 // Insert exception handling here...
 }
}
```

For callback queries (DKDatastoreICM.executeWithCallback()), the exception can be caught in the callback object's reportException method. See the following Java code snippet in the callback object.

```
public void reportException(DKException exc) {
 // Check for query timeout exception
 if (exc.getErrorId() == DKMessageIdICM.DK_ICM_MSG_QUERY_TIMEOUT) {
 // Insert exception handling for timeout here...
 } else {
 // Insert exception handling here...
 }
}
```

---

## Java viewer does not support Excel documents that have document protection enabled

### Symptom

If you have document protection enabled in an Excel document, the document will not display properly.

## Possible Causes

The Java viewer does not support data protection and encryption, a feature available in some Microsoft Office applications.

## Action

To display the document, you must disable the document protection feature in Excel.

---

## Java viewer: UTF-8 and UTF-16 plain text documents always appear left-aligned

### Symptom

The Java viewer supports viewing of UTF-8 and UTF-16 plain text documents. But these documents will always be left-aligned, even if the entire text of the document is in a language that is right-aligned, such as Arabic and Hebrew.

### Action

You can work around the problem by using one of the following options:

- Use Hebrew or Arabic encoding instead of UTF-8 for documents that you know are in Hebrew or Arabic.
- Use bidirectional marks within the UTF-8 document to point out the lines that should be read from right to left. Note that with this approach the line will still be left-aligned, but it will be read from right to left.

---

## Java viewer: Line wrapping causes incorrectly positioned graphical annotations relative to the text

### Symptom

The line breaks between word boundaries change the positioning of text on pages where the lines wrap. The graphical annotations are incorrectly positioned relative to the text. This is especially evident with highlighted annotations.

### Action

There are two possible workarounds. If very few plain text documents are annotated or if plain text documents rarely have wrapping lines, there will be few cases of annotation misalignment. Any cases that are encountered can be fixed manually. If this is not practical, the best alternative solution is to convert existing plain text documents with annotations to TIFF by using the DB2 Content Manager V8.3 Fix Pack 1 version of the Java viewer toolkit.





---

## 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.

Portions of this product are:

- Copyright © 1994-2007 EMC Corporation. All Rights Reserved

- Oracle® Outside In Viewer Technology, Copyright © 1992, 2007, Oracle. All Rights Reserved.
- Document Viewer © 1991-2007 MS Technology, Inc. Charlotte, NC. All Rights Reserved.
- Copyright © 2000-2007 The Apache Software Foundation. All Rights Reserved.
- Copyright 1994-2007 W3C (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved.
- Copyright © 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian. All Rights Reserved.
- Copyright ©1998-2003 The OpenSSL Project. All Rights Reserved.

This product is Built on Eclipse™ (<http://www.eclipse.org>).

---

## Trademarks

This topic lists IBM trademarks and certain non-IBM trademarks.

See <http://www.ibm.com/legal/copytrade.shtml> for information about IBM trademarks.

The following terms are trademarks or registered trademarks of other companies:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel and Pentium® are trademarks of Intel Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.



---

## Glossary

This glossary defines terms and abbreviations specific to this system. Terms shown in *italics* are defined elsewhere in this glossary.

### A

**abstract class.** An object-oriented programming *class* that represents a concept; classes derived from it represent implementations of the concept. You cannot construct an object of an abstract class; that is, it cannot be instantiated.

**access control.** The process of ensuring that certain functions and stored *objects* can be accessed only by authorized users in authorized ways.

**access control list (ACL).** 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.

**administrative ACL.** Any *access control list (ACL)* created by an administrator with *SystemSetACL* and *SystemDefineACL privileges* and for use with administrative objects, such as *item types* and *item type*

*views*, or *items*. ACLs that you created prior to Version 8 Release 3 are all administrative ACLs.

**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*.

**CIFS.** See *Common Internet File System*.

**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.

**Common Internet File System (CIFS).** A protocol that enables collaboration on the Internet by defining a



remote file-access protocol that is compatible with the way applications already share data on local disks and network file servers.

**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<sup>®</sup> 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 administrators.** 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

**NAS.** See *network-attached storage*.

**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 (NAS).** 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 ACL.** Any *access control list (ACL)* created by an end *user* with UserACLOwner *privilege* and can be

assigned only to *items*. Users can search on user ACLs. User ACLs do not display in the system administration client. A user who is listed in the user ACL and who has UserACLOwner privilege, or an administrator, can modify a user ACL by using the APIs.

**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

## Special characters

.NET 1.1 client 422

.NET 2.0 client 423

## A

access control 143

- assigning a list to an item 128
- Content Manager Version 8 70
- defining lists (ACLs) 125
- diagram 70
- lists (ACL) 70, 74
- NoAccessACL 74
- PublicReadACL 74
- retrieving and displaying lists (ACLs) 127
- rules for lists (ACLs) 74
- SuperUserACL 74
- working with 122
- working with lists for document routing 234

ACL 28

ACL configuration 77

ActionListName 383

actions

- accessing lists 279
- creating 279
- DKWorkflowActionICM 213
- listing in XML 391
- predefined 391
- workflow 210

ad hoc routing 231

AddItemToFolderRequest 368, 407

additional parameters field,  
OnDemand 244

administration

- object
- retrieving 159
- privileges 71

administration database

- schema mapping 5

administration metadata

- importing and exporting 325

AIX

- shared objects for C++ 21
- starting the RMI server 20

AllPrivSet 71, 74

annotation scope

- GLOBAL 327
- LOCAL 327
- VIEW 327

annotations

- adding an object to an XDO 47
- beans 295
- constant 92
- customizing 458
- documents 441
- editing support 457
- engine 444
- OnDemand 254

annotations (*continued*)

- semantic type 92
- services 455
- services classes 443

API enhancement 155

API log files

- C++ 488
- cmblogconfig.properties 487
- configuring 487
- dklog.log 487
- Java 487

application building

- creating in Content Manager Version 8 83
- non-visual beans 306
- planning 77
- using annotation services 458
- visual beans 318

application programming interfaces (API)

- Java 17
- architecture 18
- differences from C++ 17
- multiple search 29
- packaging 18

application programming interfaces (APIs)

- capabilities 17
- online reference 82
- software components 83

application programming reference (APR) 82

ApplicationName 391

arithmetic query

- operations 188
- syntax 205

ASCENDING 205

attachments

- MIME 431

attributes

- accessing in a DDO 38
- beans 295, 475, 476
- comparing with DDOs and XDOs 8
- Content Manager Version 8 64
- creating in CM 8 89
- defining 243
- definition 64
- deleting for DKAny 59
- displaying in beans 306, 315
- grouping 64
- listing for an item type 92
- listing for OnDemand 245
- listing in CM for AS/400 262
- listing in IP OS/390 256
- managing groups in CM8 90
- managing in the controller servlet 482
- modifying for an item 100
- multi-valued 64
- querying 188
- reference 173
- representing through DDOs 84

attributes (*continued*)

- retrieving for a document item type 159
- retrieving properties from 40
- setting and retrieving in CM 8 97
- updating groups in CM 8 90
- user-defined 173
- making text searchable 176
- versioning 104

ATTRONLY, ImagePlus for OS/390 261

AuditComment 391

authentication

- Web services 428

authentication, XML 355

AuthenticationData 355

## B

BasicExpression 205

batching XML requests 399

BatchRequest 399, 407

BeanInfo 295

beans 295

- ancillary 295

- annotation 295

- annotations 295

- batch support 294

- BeanInfo classes 295

- building in WebSphere Studio

- Application Developer 293

- characteristics

- customization 291

- events 291

- introspection 291

- methods 291

- persistence 291

- properties 291

- CMBSchemaManagement 293

- connection 293, 474

- connection pool

- CMBConnectionPool 474

- data management 293, 474

- data source

- datasource bean 474

- document services 295, 474

- documents 478

- event and listener classes 295

- exception classes 295

- folders 477

- helper 295

- introduction 291

- invoking 293

- items 474

- JAR files 293

- Java Viewer related

- definition 291

- locale, changing 304

- message schema 352

- Model View Controller (MVC) 295

- non-visual

- building applications 306



- beans (*continued*)
  - non-visual (*continued*)
    - categories 295
    - configurations 295
    - considerations 300
    - definition 291
    - events 305
    - features 295
    - introduction 294
    - properties 305
  - other builders 292
  - query service 474
  - requirements 292
  - schema 293
  - schema management 474
  - search criteria 475
  - search request 293
  - search results 474
  - search template 474, 475
  - searching OnDemand 254
  - session listeners 295
  - singletons 300
  - trace log 474
  - tracing 305
  - understanding basic concepts 291
  - user management 474
  - using in builders 292
  - versioning 306, 316
  - visual
    - attributes editor 315
    - building applications 318
    - collation strength 317
    - connecting 318
    - connection 317
    - default viewers 315
    - definition 291
    - document viewer 313
    - external viewers 315
    - folder viewer 312
    - general behaviors 317
    - help events 317
    - hiding and showing buttons 317
    - introduction 306
    - key events 318
    - login panel 306
    - names 306
    - overriding pop-up menus 312
    - pop-up menus 311
    - replacing 318
    - save/restore configuration 317
    - search panel 310
    - search results viewer 311
    - search template list 308
    - search template viewer 309
    - specialized behaviors 318
    - text search areas 309
    - tree pane 311
    - using in windows 320
    - versions viewer 316
    - viewer specifications 314
  - web.xml 293
  - workflow 293, 295
  - worklist 293
  - XML 295
  - XML services 352
- BETWEEN 195, 205
- binary data, 7

- binary large object (BLOB)
  - classes 243
  - creating XDOs for 42
- BLOB, 42
- buffer, adding an XDO 45
- business application node 210
- Business Process Choreographer 417

## C

- C# samples 417
- C++
  - constants 31
  - DKAny 54
  - DKConstant.h 17
  - DLL files 21
  - escape sequences 199
  - libraries 21
  - setting up the environment 21
  - shared objects for AIX 21
  - XML support 17
- C++ API 27, 28, 153
- caching, controller servlet 480
- callback, executing 29, 171
- changePassword 85
- ChangePasswordRequest 356, 407
- checked out item
  - querying 173
  - querying by person 188
  - querying by timestamp 188
  - querying example 188
- checkin 371
- CheckinItemRequest 378
- CheckInItemRequest 407
- checkout 371
- CheckoutItemRequest 378
- CheckOutItemRequest 407
- child components
  - Content Manager Version 8 65
  - creating in CM 8 92
  - diagram 65
  - referencing 67
  - representation in the data model 173
  - representing as DDO attributes 84
  - versioning 69
- class name terminology 243
- cmadmin.xsd 321
- cmb81.jar 292
- CMBAnnotationPropertiesInterface 458
- CMBAttribute
  - changing locale in display names 304
- cmbcc2mime.ini 480
- cmbclient.ini 480
- cmbcm81.lib 21
- cmbcm81d.lib 21
- CMBConnection 254, 293, 474
- CMBConnectionPool bean 301
- cmbcs.ini 295, 480
- CMBDataManagement 293, 474
- CMBDocumentServices 446, 474
- CMBDocumentViewer 306, 313
- specifications 314
- terminating 313
- CMBEntity
  - changing locale in display names 304
- CMBFolderViewer 306, 312
- CMBGenericWebService 425

- CMBGenericWebService (*continued*)
  - integrating 425
- CMBItem 474
- CMBItemAttributesEditor 315
- CMBLoginPanel 306
- cmbmessages.xsd 352
- CMBObject 476
- CMBPage 478
- CMBPageAnnotation 457
- CMBQueryService 293, 474
- cmbregist81.bat 20
- cmbregist81.sh 20
- CMBSchemaManagement 474
- CMBSearchPanel 310
- CMBSearchResults 474
- CMBSearchResultsView 254
- CMBSearchResultsViewer 306, 311
- CMBSearchTemplate 474
- CMBSearchTemplateList 254, 306, 308
- CMBSearchTemplateViewer 254, 306, 309
- cmbservlet.properties 478
- CMBServletAction 478
- cmbservletjsp.properties 478
- CMBSpecificWebService.cs 421
- CMBSTCriterion 475
- cmbsvclient.ini 480
- cmbsvcs.ini 480
- CMBTraceLog 474
- CMBUserManagement 474
- cmbview81.jar 447
- CMBViewerConfiguration.properties 447
- CMBXMLAttachment 295
- CMBXMLMessage 295
- CMBXMLMessage bean 352
- cmbxmldata.xsd 321
- CMBXMLServices 295
- cmdatamodel.xsd 321
- CMWebService
  - integrating 417
- CMWebService Web service 420, 421
- CMWebServiceClient.cs 424
- CMWebServiceClient.readme 437
- CMWebServiceClientWSE30.readme 437
- COBRA
  - Persistent Data Service (PDS) 6
  - Persistent Object Service 6
- code page conversion
  - setting console subsystem in Windows 23
- collection point
  - defining 219
  - description 209
- CollectionResumeListEntry 383
- collections
  - deleting 59
  - federated 60
- collections and iterators
  - C++
    - memory management 59
  - Java 57
    - sequential collection 57
    - sequential iterator 58
- com.ibm.mm.sdk.client 18
- com.ibm.mm.sdk.server 18
- combined query
  - definition 29

- Combined query
  - Java 183
- CompareOperator 205
- Comparison 205
- configString 355
- configuring JDBC 414
- connection bean 474
- connection pool 301
- connection pooling 415
  - servlet 478
  - servlet values 480
- connector logging 487
  - C++ 488
  - Java 487
- connectors
  - controller servlet values 480
- connectString 355
- connectToWorkflow 355
- console subsystem, setting in
  - Windows 23
- constants 31
  - routing 235
- container variables, workflow 210
- ContainerData 389
- ContainerDefinition 383
- containers
  - constant 92
  - Content Manager Version 8 67
  - semantic type 92
- contains-text-basic, text search 176
- contains-text-db2
  - query example 188
- contains-text, text search 177
- CONTENT
  - ImagePlus for OS/390 261
- Content Manager for AS/400
  - index classes 262
  - introduction 262
  - listing entities and attributes 262
  - mapping from federated 9
  - running a query 264
- Content Manager Version 8
  - access control 70
  - attributes 64
  - basic concepts 64
  - checking items in and out 109
  - child components 65
  - components illustration 63
  - connecting to 84
  - controlling access
    - access lists 74
    - user groups 73
  - controlling access privileges 71
  - creating a content server 84
  - creating a document 161
  - creating a password 85
  - creating an application 83
  - creating attributes 89
  - creating item types 85
  - defining a resource item type 101
  - deleting a document 164
  - document parts 65
  - documents 65, 69
  - folders 69
  - insurance scenario sample 83
  - introduction 63
  - item types 65

- Content Manager Version 8 (*continued*)
  - items 64
  - library server 63
  - library server and resource manager
    - consistency 165
  - linking items 120
  - links 67
  - listing attributes for an item type 92
  - listing item types 87
  - managing attribute groups 90
  - managing documents 158
  - mapping from federated 9
  - modifying item attributes 100
  - non-resource items 65
  - objects 66
  - planning an application 77
  - privilege set 71
  - query examples 184
  - querying 171
  - references 67
  - resource items 65
  - resource manager 63
  - retrieving a document 164
  - retrieving items 104
  - root components 65
  - routing documents 209
  - samples 82
  - searching for items 103
  - setting and retrieving item
    - attributes 97
- System Managed Storage (SMS)
  - server 63
- transactions 165
- understanding the query
  - language 146
- understanding the search query 171
- updating documents 163
- versioning 69
- working with access control 122
- working with folders 113
- working with the resource
  - manager 147

- Content Manager, earlier versions
  - mapping from federated 9
- content parts
  - attaching in Web services 431
- content servers
  - accessing information on 243
  - beans 295
  - class names 243
  - Content Manager for AS/400 262
  - Content Manager Version 8.4 63
  - controller servlet values 480
  - creating 243
  - data definition hierarchy 243
  - defining 243
  - dynamic data object (DDO) 6
  - extended data object (XDO) 7
  - federated 9
  - ImagePlus for OS/390 256
  - managing data 5
  - OnDemand 244
  - persistent identifiers (PIDs) 8
  - registering in federated 10
  - relationship to DDOs 7
  - result set cursor 243

- content servers (*continued*)
  - retrieving the administration
    - object 159
  - servers supported by EIP 5
  - using with RMI 20
- content storage 153
- contentOption 360
- ContinueProcessRequest 396, 407
- controlled entities 70
- controller server
  - defaults 480
- controller servlet
  - actions 478
  - clean up 478
  - connection pooling 478
  - connection pooling values 480
  - conventions 480
  - conversion parameters 480
  - extending 478
  - JSP sets 478
  - locale 478
  - names separator parameter 480
  - pages that show error messages 480
  - parameters 480, 482
  - properties file 478, 482
  - reference 479
  - replay 478
  - request parameters
    - action 482
    - connection related 482
    - document related 482
    - folders 482
    - general 482
    - items 482
    - managing content 482
    - reply 482
    - search 482
  - service runtimes 480
  - session management 478
  - toolkit function matrix 484
  - tracing 480
  - using 478
- conversions 459
- convert, schemas 350
- createChildDDO 92
- createDDO 36
- CreateItemRequest 358, 407
- CreateLinkRequest 380, 407
- creating 92
- criterion, search, bean 475
- cs package 18

## D

- data definition classes 243
- data instances
  - importing and exporting 345
- data items, DDO 38
- data model
  - applying the query language 173
  - items
    - representation in the data
      - model 173
    - representing in XML 324
    - representing query in XML 184
- data model objects
  - importing and exporting 327

- data\_id 40
- databaseNameStr 84
- datastore, 5
- DB2
  - client configuration assistant 21
  - client support 21
- DB2 connector 4
- DB2 Content Manager V8 4
- DB2 Extenders 7
- DD connector (Domino Doc 4
- ddoDocument 47
- decision point 210
- deleteDKAttributeDef 59
- DeleteItemRequest 377, 407
- DeleteLinkRequest 380, 407
- deleteSchemaMapping 350
- DemoSimpleAppl.java 306
- deprecated features 4
- DESCENDING 205
- DfltACLCode 74
- diagnostic information, 29
- DIGIT 205
- display flag, foreign key 97
- display names
  - changing the language of 304
- DisplayName 391
- displayNamesEnabled 304
- DIV 205
- DK\_CM\_OPT\_ACCESS\_MODE 243
- DK\_CM\_READWRITE 243
- DK\_CM\_VERSION\_LATEST 104
- dkAbstractWorkFlowUserExit 279
- DKAny
  - assignment from 56
  - assignment to 55
  - checking the type 56
  - deleting collections 59
  - destroying 56
  - display of 56
  - memory management 55
  - programming tips 57
  - type code, getting 55
  - Typecode 55
  - using inside a collection 57
  - using type constructors 55
- dkAttrDef 243
- DKAttrGroupDefICM 90
- dkBlob 243
- DKCollectionResumeListEntryICM 213
- DKConstant 31
- DKConstant.h 17
- DKDatastore 243
- dkDatastoreDef 243
- DKDatastoreDL
  - Java 31
    - connecting 32
    - DKDatastoreDL options 33
    - list schema and schema attributes 34
    - List servers 33
- DKDatastoreICM 83
- DKDatastoreIP 256
- DKDatastoreOD 244
  - listEntities 245
- DKDatastoreV4 262
- DKDatastorexx 243
- DKDatastoreIP
  - listEntityAttrs 256
- DKDatastoreOD
  - listSearchTemplates 247
- DKDDO, 36
- DKDocRoutingServiceICM 213
- DKDocRoutingServiceMgmtICM 213
- DKDocumentConverter 350
- dkEntityDef 243
- DkEntityDefIP
  - getAttr 256
- DKEntityDefIP
  - listAttrNames 256
- DKException 498
- DKException 29, 78
- DKFederatedCollection 60
- dkFederatedIterator 60
- dkFederatedQuery 60
- DKFederatedQuery 11
- DKFixedView 244
- DKFixedViewDataOD 244
- dkIterator 60
- DKLink 120
- DKLinkCollection 121
- DKPrivilegeSetICM 124
- DKProcessICM 213
- DKResource 244
- DKResourceGrpOD 244
- DKResults
  - positioning the iterator 60
- dkResultSetCursor 243
- DKResumeListEntryICM 213
- DKRMConfiguration 84
- DKRMConstantSMS 155
- DKRMControlServiceSMS 155
- DKRMServerDefinitionDefSMS 155
- DKRMVolumeDefSMS 155
- DKRouteListEntryICM 213
- DKSchemaConverter 350
- dkServerDef 243
- DKTimestamp
  - suspending a workflow 274
- DkViewOD 244
- DKViews 244
- DKWorkFlowActionICM 213
- DKWorkFlowFed
  - resume 275
  - suspend 274
- DKWorkFlowServiceFed
  - svWf 273
- DKWorkFlowServicesFed 271
- dkWorkFlowUserExit 279
- DKWorkItemFed
  - checkIn 278
  - checkOut 278
- DKWorkListFed 276
- DKWorkListICM 213, 223
- DKWorkNodeContainerDefICM 213
- DKWorkNodeICM 213, 218
- DKWorkPackageICM 213, 229
- DKXMLExportList 324
- DKXMLSysAdminService
  - constants 324
  - methods 324
- DL connector (Content Manager version 7) 4
- DLL files
  - C++ 21
- DLLName 391
- doAction 279
- DOCROUTINGFREQ column
  - changing 212
- Document and encoded 404
- Document and literal 404
- document model, 65
- Document Object Model (DOM)
  - building from an XML message 426
- document parts
  - constant 85
  - Content Manager Version 8 65
  - item type 85
- document routing 132
- document routing, 209
- document types 459
- document viewer toolkit
  - annotation services 455
  - applet or servlet 446
  - architecture 443
  - creating a generic viewer 447
  - customizing the generic document viewer 447
  - dual-mode and applet or servlet 447
  - engines
    - MS-Tech Document 444
  - example applications 444
  - introduction 441
  - Java application 445
  - page manipulation 461
  - standalone viewer 445
  - thin client 446
- documents 92
  - annotating 441
  - beans 295, 474, 478
  - caching in the controller servlet 480
  - Content Manager Version 8 65, 69
  - creating a document item type 159
  - creating in CM 8 161
  - creating the management data model in CM 8 159
  - deleting in CM 8
    - SDocModelItemICM 164
  - displaying in beans 306, 313
  - managing in CM 8 158
  - representing in the data model 173
  - retrieving in CM 8 164
  - routing through a process 209
  - semantic type 92
  - starting a routing process 225
  - text searching for 176
  - updating in CM 8 163
  - versioning parts in the management data model 165
  - viewing, 441
- dropped features 4
- dsType 254
- dynamic configuration, beans 295
- dynamic data object (DDO)
  - accessing attributes 38
  - accessing folder contents 52
  - adding properties 37
  - attribute groups in CM 8 90
  - COBRA specification 6
  - comparing with attributes 8

- dynamic data object (DDO) (*continued*)
  - federated 9
  - introduction 6
  - obtaining all folders containing 118
  - persistent identifier 8
  - populating with setData 92
  - referencing 67
  - relationship with content servers 7
  - representing multimedia content 7
  - retrieveObject 104
  - retrieving attribute properties 40
- Dynamic Data Object (DDO)
  - C++
    - deleting 42
  - Java 36
    - adding 38
    - attribute, DKPARTS 49, 52
    - creating 36
    - data item values 38
    - displaying 41
    - Information, Digital Library 84
    - PID 37
    - properties 40

## E

- EIP workflow services, 271
- Enterprise Information Portal
  - databases 5
  - document engines 444
  - dynamic data objects (DDO) 6
  - extended data objects (XDOs) 7
  - persistent identifier (PID) 8
  - schema mapping 5
  - supported content servers 5
- Enterprise Java Beans (EJBs) 294
- EnterUserDLL 383
- EnterUserFunction 383
- entities
  - beans 295, 475
  - class hierarchy 243
  - controlled, Content Manager Version 8 70
  - identifying 8
  - ImagePlus for OS/390 261
  - listing in CM 8 87
  - listing in CM for AS/400 262
  - listing in IP OS/390 256
  - listing in XML 357
  - mapping in federated 9
  - moving items through XML 382
  - native 254
  - OnDemand folders as native 254
  - privileges 71
  - storing federated folders in 14
- ENTITY\_TYPE 244, 253
- EntityList 357
- environment, setting
  - C++ 21
  - Java 19
- ErrorCode 78
- ErrorState 78
- escape sequences, query examples 194
- ESCAPE\_LITERAL 205
- ESCAPE\_KEYWORD, query 205
- evaluating queries 61
- evaluating search 29

- events
  - beans 295
- EXCEPT 184, 205
- exceptions, handling 29
- executing queries 61
- executing search 29
- executing search with callback 29, 171
- exponent, query 205
- exporting XML 324
- expression, query 205
- ExpressionList 205
- expressions, query 188
- ExpressionWithOptionalSortBy 205
- extended data object, 7
- Extended Search
  - mapping from federated 9
- Extenders 7
- extensible markup language, 17
- extracting XML
  - administration metadata 325
  - data instances 345, 346
  - data model metadata 327
  - exporting object dependencies 349
  - input and output formats 324

## F

- Fed connector 4
- federated 14
  - attributes 9
  - beans 475
  - collection 11
  - collections 60
  - document model 9
  - entities 9
  - folders 14
  - image search 12
  - items 9
  - iterators 60
  - layer for beans 295
  - mapping 9
  - mapping user IDs 10
  - query
    - creating 11
    - examples 12
    - processing illustration 11
    - results 11
    - syntax 12
  - registering content servers 10
  - schema mapping 10
  - searching
    - illustration 9
    - introduction 9
    - process 11
    - relational illustration 11
  - system administration functions 15
- federated document model 9
- fetching elements 28
- file, adding an XDO from 46
- FLOAT\_LITERAL 205
- folders
  - accessing 52
  - adding contents to 114
  - beans 477
  - Content Manager behavior 52
  - Content Manager Version 8 69
  - creating in CM 8 114

- folders (*continued*)
  - displaying in beans 306, 312
  - enabling the mode in
    - OnDemand 253
  - listing in OnDemand 247
  - managing in the controller
    - servlet 482
  - obtaining all folders with a specific
    - DDO 118
  - OnDemand folders as native
    - entities 254
  - removing contents from 115
  - retrieving contents 117
  - semantic type 92
  - XML 368
- foreign key attribute values 97
- fromXML() 345
- FunctionName 205, 391

## G

- garbage collector, Java 17
- generic document viewer, 441
- GenericWebServiceSample.java 425
- GenericWebServiceSample.readme 437
- getDataModelDefs 324
- getDisplayName() 304
- getLocale 304
- getName() 304
- getNonDisplayName() 304
- GetPrivilegesRequest 367, 407
- getSchemaMappingNames 350
- getStorageSchema 350
- getSysAdminDefs 324
- getXSLTQuery 350
- GLOBAL 327
- graphical user interface (GUI), 291

## H

- helper beans 295
- Hierarchical Storage Management (HSM) 147
- history
  - constant 92
  - semantic type 92

## I

- IBM® DB2® Driver for JDBC and SQLJ 80
- ICM connector, 63
- ICMAclPrivExit 143
- ICMBASE
  - representing in Web services 431
- ICMCHECKEDOUT
  - attributes 173
  - example 188
- ICMCHKOUTTS 188
- ICMCHKOUTUSER 188
- ICMFetchContentexit 133
- ICMGenPrivExit 143, 145
- ICMLogon 71
- ICMLogonExit 136
- ICMRMPREFETCH table 237

- ICMSTSYSCONTROL table
  - configuring 212
- ICMValidatePassword 140
- IDENTIFIER 205
- Image Plus (IP/390) 4
- image search
  - definition 29
  - federated 12
- ImagePlus for OS/390
  - introduction 256
  - listing attributes 256
  - listing entities 256
  - mapping from federated 9
  - query parameters 261
  - query syntax 261
- importing XML 324
- inbound links 120
- INBOUNDLINK 173
- Information Catalog
  - mapping from federated 9
- information center 82
- Information Mining
  - beans support 295
- ingesting XML
  - administration metadata 325
  - constants 324
  - data instances 345, 348
  - data model metadata 327
  - input and output formats 324
- initialization parameters field,
  - OnDemand 244
- INSO stop 178
- INTEGER\_LITERAL 205
- INTERSECT 184, 205
- IPv6 support 79
- IS 205
- item classes, 65
- item parts
  - comparing with DDOs and XDOs 8
- item types
  - classifications 85
  - components, root and child 65
  - Content Manager Version 8 65
  - creating in Content Manger Version 8 85
  - importing and exporting as XML
    - schemas 327
  - listing 87
  - listing attributes for 92
  - querying multiple 188
  - versioning 69, 111
- ItemAdd 71, 232
- ItemAdminPrivSet 71
- ItemDelete 232
- ItemQuery 71
- ItemReadPrivSet 74
- items
  - adding to a folder 114
  - beans 295, 474, 476
  - checking in and out 109
  - Content Manager Version 8 64
  - creating a document item type 159
  - federated 9
  - importing and exporting instances as XML 345
  - linking 120
  - linking through folders 69

- items (*continued*)
  - managing with the controller
    - servlet 482
  - modifying attributes 100
  - referencing 84
  - removing from folders 115
  - retrieving in CM 8 104
  - retrieving linked items 121
  - searching for in CM 8 103
  - setting and retrieving attributes in CM 8 97
  - tree constant 117
  - versioning 69, 111
- ItemSetSysAttr 71
- ItemSetUserAttr 71
- ItemSetUserAttrs 232
- ItemSQLSelect 71
- ItemSuperAccess 232
- ITEMTREE 360
- ItemTypeQuery 71
- iterators
  - federated 60
  - positioning in DKResults 60

## J

- j2ee.jar 293
- Java
  - constants 31
  - creating workflow actions 279
  - DKConstant.h 17
  - document viewer toolkit 441
  - escape sequences 199
  - garbage collector 17
  - increasing JVM stack size 253
  - packages 18
  - setting up the environment 19
  - XML functions 17
- Java API 28, 153
- Java Server Pages (JSPs)
  - servlet 478
- Java viewer
  - compression selection 464
  - document types 459
  - engine property settings 464
  - preferences 465
- java.lang.exception 78
- java.lang.objects 97
- JavaBeans, 291
- JAX-RPC based client toolkit 425
- JCC connector 80
- JDBC
  - configuring 414
- JDBC connector 4
- JNI tracing, servlet 480
- joining routes 210
- JSPs, 473

## L

- large object (LOB) 84
- LeaveUserDLL 383
- LeaveUserFunction 383
- LETTER 205
- library
  - C++ listing 21

- library (*continued*)
  - Java 19, 21
- library server 132
- library server, Content Manager Version 8 63
  - listing 84
  - versioning 69
- LIKE 195, 205
- links
  - attribute type 84
  - Content Manager Version 8 67
  - creating in XML 380
  - defining between items 120
  - description item 120
  - inbound and outbound 120
  - representation in the data model 173
  - retrieving linked items 121
  - source 120
  - target 120
  - traversing through query 188
  - type name constants 121
  - type names 121
- LinkTypeName 120
- ListActionNamesRequest 391, 407
- ListActionRequest 407
- ListConstructor 205
- ListContent 205
- listDataSources 84
- listEntityNames 87
- ListNextWorkPackageRequest 407
- ListNextWorkPackagesRequest 389
- ListProcessNamesRequest 386, 407
- ListProcessRequest 386, 407
- listResourceMgrs 84
- ListSchemaRequest 357, 407
- ListServerRequest 354
- ListServersRequest 407
- listWorkFlowTemplates 278
- ListWorkListNamesRequest 387, 407
- listWorkLists 275
- ListWorkListsRequest 387, 407
- ListWorkNodeRequest 407
- ListWorkNodesRequest 383, 407
- ListWorkPackagesRequest 389
- literal, query 205
- literals
  - ImagePlus for OS/390 261
- literals, query 188
- LOCAL 327
- locale
  - beans, changing in 304
  - servlet 478
- LocalPart 205
- logging 415
- logging configuration file
  - APIs 488
  - modifying 488
  - setting values 488
- logging, 29
- LogicalOrSetExpression 205
- LogicalOrSetPrimitive 205
- LoginData 355
- LorgicalOrSetTerm 205
- LTPA tokens 435



## M

- maxResults 360
- Message Translation Optimization Mechanism (MTOM) 431
- messages
  - DKException information 78
- Microsoft .NET Studio 2003 404
- Microsoft .NET Studio 2005 404
- Microsoft SOAP Toolkit 421
- Microsoft Visual C++ compiler, 21
- Microsoft Visual Studio .NET
  - getting started in Web Services 421
- Mime2App property 314
- MimeMultipart object
  - MIME types 431
- MOD 205
- Model View Controller (MVC) 295, 457
- MoveltemRequest 382, 407
- multimedia content 7
- Multipurpose Internet Mail Extensions (MIME) 431
- multistreaming 17
- myExit (document routing user exit) 139

## N

- nameOfAttrStr 100
- NameText 205
- NewValues 382
- newVersion 371
- NoAccessACL 74
- NodeGenerator 205
- non-display names 304
- non-resource items
  - Content Manager Version 8 65
- Non-visual beans 291
- NONZERO 205
- NoPrivSet 71
- NOT 205
- note
  - constant 92
  - semantic type 92
- notelog 476
- NotifyState 389
- NotifyTime 389
- NULL, query 205

## O

- object management
  - Java
    - creating 92
    - deleting 108
    - updating 100
- Object Management Group (OMG) 6
- objects
  - beans 476
  - Content Manager Version 8 66
  - storing attribute values 97
  - text searching for contents 176
  - updating in CM 8 100
- ODBC connector 4
- OnDemand
  - annotations 254
  - connecting to 245
  - disconnecting from 245

- OnDemand (*continued*)
  - displaying attributes 250
  - displaying documents 250
  - enabling the folder mode 253
  - introduction 244
  - listing application groups 245
  - listing folders 247
  - listing server information 245
  - mapping from federated 9
  - property names 244
  - querying an application group 248
  - representing servers and documents 244
  - retrieving documents 248
  - searching asynchronously 253
  - searching for a document 248
  - tracing 254
  - using folders as search template 254
- operators
  - ImagePlus for OS/390 query 261
  - parametric search 175
- OptionalExpressionList 205
- OptionalPredicateList 205
- OR 205
- outbound links 117, 120
- output\_option 33
- OverLoadLimit 383
- OverloadUserDLL 383
- OverloadUserFunction 383

## P

- packages
  - client and server (cs) 18
  - hierarchy in Java 18
  - Java client 18
  - Java server 18
- page
  - manipulation
    - functions 461
- page manipulation
  - interfaces 462
- pages
  - beans 478
- parallel routing 210
- parametric search
  - definition 29
  - federated 12
  - operators 175
  - querying in CM for AS/400 269
  - understanding 175
- parts
  - versioning 165
- password
  - changing in beans 317
  - changing in XML 356
  - creating in Content Manager Version 8 85
  - mapping in federated 10
- password validation 132, 140
- PENDING, ImagePlus for OS/390 261
- pEnt, C++ 34
- permissions, 122
- Persistent Data Service (PDS), COBRA 6
- persistent identifier (PID)
  - dynamic data object (DDO) 6
  - extended data object (XDO) 7

- persistent identifier (PID) (*continued*)
  - introduction 8
  - routing 228
  - versioning 69
- Persistent Object Service, COBRA 6
- policies example 83
- predefined values 97
- predicate list 205
- prefetching objects 237
- priority, document routing 389
- privileges
  - access control lists (ACLs) 74
  - beans 295, 476
  - codes 71
  - Content Manager Version 8 70, 71
  - creating a set 124
  - creating in CM 8 122
  - data access 71
  - displaying properties of a set 125
  - granting for routing 232
  - pre-configured ACLs 74
  - pre-defined Content Manager Version 8 sets 71
  - set 71
  - system-defined 71
  - types of sets 71
  - users and user groups in CM 8 73
  - viewing in XML 367
- PrivSetCode 71
- ProcessCompletionTime 389
- processes
  - continuing 227
  - continuing in XML 396
  - defining for an associated route 223
  - DKProcessICM 213
  - ending 226
  - joining 210
  - listing 230
  - listing in XML 386
  - resuming 228
  - resuming in XML 398
  - routing 213
  - splitting 210
  - starting in XML 395
  - subprocesses 210
  - suspending 227, 397
- programming
  - DB2 Content Manager 3
- proxy classes 422, 423
- PublicReadACL 74

## Q

- QBIC connector (Image Search) 4
- QName 205
- queries
  - arithmetic operations 188
  - CM for AS/400 264
  - Content Manager Version 8 171
  - data model 184
  - escape sequences 194
  - evaluate 61
  - examples 184, 188
  - execute 61
  - federated
    - processing 11
  - federated collection 60

## queries (*continued*)

- ImagePlus syntax 261
- language 194
- language grammar 205
- listing checked out items 188
- listing the result 188
- OnDemand search 248
- routing examples 232
- syntax 188
- text search 176
- understanding the language 171
- version 188
- wildcard example 188
- XML items 360

## Query

- Java 61
  - dkResultSetCursor vs DKResults 62
  - parametric type 62
  - query object types 61
- query language 194
- query options 171
- query timeout 498
- query timeout option 103
- QueryCriteria 360

## R

- reference attributes 173
- REFERENCEDBY 173
- REFERENCER 173
- references
  - attribute type 84
  - Content Manager Version 8 67
  - traversing through query 188
- relational databases
  - mapping from federated 9
- remote method invocation (RMI)
  - connecting with beans 295
  - starting in Java 20
  - using with Java APIs 20
- Remote Procedure Call (RPC) 401
- removeAllElement 59
- RemoveItemFromFolderRequest 368, 407
- replaceElementAt 59
- ReplayAction, servlet 478
- resource content, 7
- resource items
  - constant 85
  - Content Manager Version 8 65
  - defining an item type for 101
  - item type 85
  - representing in Web services 431
- resource manager 155
- resource manager, Content Manager Version 8 63
  - listing 84
  - uniform resource identifier (URI) 148
  - versioning 69
  - working with 147
- resume list 213
- ResumeList 389
- ResumeProcessRequest 398, 407
- ResumeTime 389
- resuming a workflow 275

## retrieve

- considerations with row-based filtering 199
- RetrieveFolderItemsRequest 368, 407
- RetrieveFoldersForItemRequest 407
- RetrieveItemRequest 362, 407
- retrieveOption 360
- RMI server, 20
- root components
  - Content Manager Version 8 65
  - creating in CM 8 92
  - linking 67
  - referencing 67
  - representation in the data model 173
  - versioning 69
- RouteSelected 396
- routing 295
  - access control lists (ACLs) 234
  - ad hoc 231
  - beans 295
  - collection points 209
  - compatibility with Version 8.2 212
  - constants 235
  - container variables 210
  - continuing a process 227
  - creating service objects 216
  - decision point 210
  - defining a new collection point 219
  - defining a new process and associated route 223
  - defining a new regular work node 217
  - defining a worklist 221
  - ending a process 226
  - examples 232
  - granting privileges 232
  - graphical process builder 210
  - introduction 209
  - joining 210
  - listing document routing processes 230
  - listing package PIDs in a worklist 228
  - listing work nodes 218
  - listing worklists 223
  - parallel 210
  - resume list 213
  - resuming a process 228
  - retrieving work package information 229
  - setting up 213
  - splitting 210
  - starting 225
  - starting in XML 394
  - subprocesses 210
  - suspending a process 227
  - work nodes 209
  - work package 213
  - workflow action 210
  - worklist 213
  - XML 383
- row-based filter 199
- RunQueryRequest 360, 407
- runtime ACL configuration 77

## S

- sample files
  - Content Manager Version 8 82
  - SampleMessageTemplate.java 426
  - SAttributeDefinitionCreationICM 64
- scenario, insurance sample for Content Manager Version 8 83
- schema mapping
  - associating in federated 9
  - introduction 5
- schemas
  - beans 474
  - cmadmin.xsd 321
  - cmdatamodel.xsd 321
  - conversion tool 350
  - importing and exporting storage schemas 327
  - unsupported XML types in storage schemas 344
- SConnectDisconnectICM 78
- scope
  - tag library 474
  - transaction 165
- score-basic, text search 176, 177
- SDocModelItemICM 69
- SDocRoutingDefinitionCreationICM 216, 217
- search template, OnDemand
  - using folders as 254
  - viewer 253
- searching
  - API modules 83
  - beans 295, 474, 476
  - understanding the query language 171
- searchTemplate bean 475
- SelectionFilterOnNotify 387
- SelectionFilterOnOwner 387
- SelectionFilterOnSuspend 387
- SelectionOrder 387
- semantic type
  - definition 92
  - pre-defined types 92
  - user defined 92
- SequencedValue 205
- ServerType 354
- servlet, 478
- session listeners, beans 295
- sessions
  - servlet 478
- setData 92
- setLocale 304
- settings
  - C++ environment 21
  - Java environment 19
- Settings
  - C++
    - Building on NT 23
  - Java
    - Client connect and disconnect 32
    - On NT 19, 20, 22
    - Programming tips 18
  - Using sample Java applets and servlet 473
    - Java application on client 473
    - Local access 474
    - Remote access 474

- Settings (*continued*)
  - Using sample Java applets and servlet (*continued*)
    - Retrieve servlet 473
- setToFirstCollection 60
- setToLastCollection 60
- setToNextCollection 60
- setToPreviousCollection 60
- SFolderICM 69, 113, 118
- Shortcut 391
- Simple Object Access Protocol (SOAP) 401
- single sign-on
  - configuring 434
- Single sign-on 435
- SItemCreationICM 69
- SItemDeletionICM 108
- SItemRetrievalICM 104
- SItemTypeCreationICM 65
- SItemTypeRetrievalICM 111
- SItemUpdateICM 109
- SLinksICM 67, 120, 121
- SLinkTypeDefinitionCreationICM 121
- SOAP 404
- software prerequisites 410
- SORTBY 205
- SortSpec 205
- SortSpecList 205
- SOURCEITEMREF 173
- splitting routes 210
- SReferenceAttrDefCreationICM 67, 173
- SResourceItemMimeTypeTypesICM 102
- SSearchICM 103
- stack trace 78
- StartProcessRequest 394
- Step, query 205
- storage schemas
  - importing and exporting 327
  - unsupported XML types 344
- streaming doc services 443
- STRING\_LITERAL 205
- subprocesses 210
- SuperUserACL 74
- suspending a process 227
- suspending a workflow 274
- SuspendProcessRequest 397, 407
- SuspendState 389
- svWF 273
- SYSREFERENCEATTRS 173
- system administration client
  - creating worklists 276
  - customizing 15
  - user exits 15
- System Managed Storage (SMS), Content Manager Version 8 63
- SystemAdmin 71
- SystemAdminPrivSet 71
- SystemDefineItemType 71

## T

- table definition 239
- tag library
  - connection related 474
  - document related 478
  - folder related 477
  - item related 476

- tag library (*continued*)
  - schema related 475
  - search related 476
- taglib.tld 293
- TARGETITEMREF 188
- TCallbackOD 253
- templates
  - workflow 278
- TerminateProcessRequest 395, 407
- text search
  - advanced search 177
  - basic query 176
  - beans 309
  - contains-text 177
  - DB2 Net Search Engine (NSE) 29
  - definition 29
  - making user-defined attributes searchable 176
  - OnDemand 244
  - operators 176
  - query example 188
  - query syntax 176
  - score-basic 177
  - searching for documents 176
  - searching for object contents 176
  - understanding 175
  - wildcards 176
- thin client 446
- TImageAnnotation 458
- timed-out sessions, servlet 478
- TimeLastMoved 389
- TimeLimit 383
- timing out 27
- tools
  - Content Manager Version 8 82
- toXML() 345
- tracing 415
  - beans 305, 474
  - controller servlet 480
  - error state 78
  - handling errors with DKException 78
  - JNI, servlet 480
  - OnDemand 254
  - return code container 78
  - sample tools 78
  - stack 78
- transactions, Content Manager Version 8 165
  - caution about explicit transactions 167
  - checking in and out 167
  - considerations 166
  - list 167
  - processing 167
- transformXMLDocument 350
- TRetrieveFolderWithCallbackOD 253
- TRetrieveWithCallbackOD 253
- TS connector (Text Search) 4

## U

- un-installing 416
- UNICODE\_CHARACTER 205
- UNION 184, 205
- UpdateItemRequest 371, 407
- updateObject 100

- UpdateWorkPackageRequest 407
  - XML messages
    - updating work packages 393
- updating content
  - XDO 48
- user exit 132
- user exit routines 143
- user exits
  - system administration 15
  - workflow 279
- user ID
  - mapping in federated 10
- user privileges
  - Content Manager Version 8 70, 71
- user-defined attributes 173
  - making text searchable 176
- user-specified location 153
- UserLastMoved 389
- users
  - controlling access in CM 8 73

## V

- valueObj 100
- version attribute 360
- version, querying 188
- versioning
  - application-controlled 69
  - attributes 104
  - beans 306, 316
  - Content Manager Version 8 69
  - item types 111
  - items 104
  - parts in the document management
    - data model 165
  - persistent identifier (PID) 69
  - policy 69
  - setting for an item 111
  - version control policies 111
- VideoCharger 63
- VIEW 327
- viewer preferences 465
- viewers, Java document, 441
- Visual C++ compiler 21

## W

- W3C XML Query 171
- Web services
  - advantages 401
  - architecture 403
  - authenticating 428
  - C# walkthrough sample 421
  - comparison of interfaces 404
  - configuration 414
  - configuring 414
  - connection pooling 415
  - creating a new instance 429
  - Document processing 403
  - getting started in .NET 421
  - getting started in a Java environment 425
  - getting the WSDL file 425
  - implementation 403
  - installation 411
  - integrating with applications 417



- Web services (*continued*)
  - introduction 401, 402
  - Java samples 417
  - Java template listing 426
  - Java walkthrough sample 425
  - overview 401
  - performance 436
  - post installation 414
  - programming in .NET 424
  - relation to other XML services 321
  - samples 437
  - samples location 417
  - scenario 402
  - security 428
  - standards 401
  - starting, stopping, verifying 413
  - supported platforms 410
  - supported WAS topologies 412
  - walkthrough sample 424
- Web Services Description Language (WSDL) 401
- Web services functions 406
- Web services requests 407
- web.xml 293
- WebSphere Application Server
  - authenticating 355
- Websphere Application Server
  - topologies 412
- WebSphere Studio Application Developer (WSAD)
  - building beans 292
  - creating a Web services client with 421
  - diagram 321
- wildcards
  - query 195
  - text search 176
- Windows
  - C++ DLL files 21
  - setting the console subsystem 23
  - starting the RMI server 20
- work items
  - accessing 277
- work nodes
  - business application 210
  - DKWorkNodeContainerDefICM 213
  - extending 383
  - listing 218
  - listing in XML 383
  - types 383
- work note
  - defining 217
  - routing 209
- work packages
  - DKWorkPackageICM 213
  - listing in XML 389
  - listing PID strings in a worklist 228
  - listing the next one in XML 389
  - retrieving information 229
  - updating in XML 393
- work packet
  - introduction 271
- workflow
  - accessing work items 277
  - accessing worklists 276
  - actions 210
  - beans 293

- workflow (*continued*)
  - beans support 295
  - connecting to 271
  - container variables 210
  - creating Java actions 279
  - deleting 273
  - disconnecting from 271
  - introduction 271
  - listing 273
  - listing all the templates 278
  - listing all the worklists 275
  - moving items 278
  - sending results from a federated folder 14
  - starting 272
  - suspending 274
  - terminating 273
- WorkListName 389
- worklists
  - accessing 276
  - accessing work items 277
  - beans 293
  - defining 221
  - DKWorkListICM 213
  - listing 223, 275
  - listing in XML 387
  - listing package PIDs 228
  - moving items 278
- WorkPackageOwner 389
- WorkPackagesToReturn 387
- WSDL 420, 421
  - from command line 421
  - from system administration client 420
- WSDL file 425
- wsdl.exe
  - getting started with 421
- WSDL2Java 425

## X

- XDO
  - adding an annotation object to 47
  - adding from a file 46
  - adding from the buffer 45
  - class type hierarchy 102
  - comparing with attributes 8
  - creating a resource item 102
  - deleting 48
  - examples 48
  - introduction 7
- Java 42
  - DDO, part of 44
  - PID 43
  - Programming tips 43
  - stand-alone 45
- large object 84
- objects 66
- representing multimedia content 7
- retrieving 48
- text searching 188
- updating 48

XML

- administration metadata 325
- beans 295
- beans tag library 293
- document routing 383

XML (*continued*)

- extracting
  - buffer 349
  - file 349
  - Web address 349
- item types 327
- items 345
- Java support 17
- representation of the query examples
  - data model 184
- representing metadata 345
- schema conversion tool 350
- services 321
- W3C query 171

XML beans 403

XML messages

- authenticating 355
- batching 399
- changing a password 356
- checking items out and in 378
- continuing a process 396
- creating an item 358
- deleting items 377
- ending a process 395
- linking items 380
- listing actions 391
- listing entities 357
- listing processes 386
- listing servers 354
- listing work nodes 383
- listing work packages 389
- listing worklists 387
- moving items between entities 382
- relation to Web services 403
- ResumeListDefinition 397
- resuming a process 398
- retrieving items 362
- searching items 360
- SOAP 401
- starting a process 394
- suspending a process 397
- updating items 371
- viewing user privileges 367
- working with folders 368

XML services

- diagram 321
- exporting object dependencies 349
- importing and exporting metadata 324
- overview 321
- programming layers 321
- programming with JavaBeans 352
- scenarios 324
- unsupported XML types 344

XMLItem 345

XQuery Path Expressions (XQPE) 171

XSD files 327





Program Number: 5724-B19  
5697-H60

SC27-1347-07

