IBM Content Manager Enterprise Edition
IBM Content Manager for z/OS
Version 8.4.3.1

Application Programming Guide

SC27-1347-12

IBM Content Manager Enterprise Edition
IBM Content Manager for z/OS
Version 8.4.3.1

Application Programming Guide

SC27-1347-12

Note

Before using this information and the product it supports, read the information in "Notices" on page 669.

This edition applies to version 8, release 4, modification 3, fix pack 1 of IBM Content Manager Enterprise Edition (product number 5724-B19) and version 8, release 4, modification 3, fix pack 1 of IBM Content Manager for z/OS (product number 5697-H60) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

ibm.com and related resources

Product support and documentation are available from ibm.com®.

Support and assistance

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

Content Manager EE
: http://www.ibm.com/software/data/cm/cmgr/mp/edition-enterprise.html

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
Content Manager EE	http://www.ibm.com/support/docview.wss?rs=86 &uid=swg27015910
Content Manager for z/OS	http://www.ibm.com/support/docview.wss?rs=119 &uid=swg27015911

"How to send your comments"
"Contacting IBM" on page x

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.

Consumability survey

You are invited to tell IBM how to improve the consumability of software products. If you want to help IBM make Content Manager EE easier to use, take the Consumability Survey at http://www.ibm.com/software/data/info/consumability-survey/.

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/.

Getting started with programming Content Management applications

IBM® Content Manager provides Java, JavaBeans, and C++ application programming interfaces (APIs) with IBM 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 397	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 9	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 IBM Content Manager Version 8.4" on page 59	Use the ICM connector APIs to build and deploy custom applications that access an IBM Content Manager content server. Use the APIs to integrate your existing applications into an IBM 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 205	Search for data by using parametric and text search.	Perform when you search for data by using text and parametric search.
"Routing a document through a process" on page 251	Route documents through a business process by using document routing APIs.	Perform when routing a document through worknodes, worklists, and processes.
"Understanding prefetching in IBM Content Manager for z/OS" on page 363	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 369	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 IBM Content Manager for AS/400® content servers.
"Building IBM Information Integrator for Content workflow applications (deprecated)" on page 405	Create or extend your own applications to use the IBM Information Integrator for Content workflow support by using the IBM Information Integrator for Content for Content classes and APIs.	Perform when using IBM Information Integrator for Content classes and APIs.
"Building applications with non-visual and visual JavaBeans" on page 425	Use the visual and non-visual JavaBeans™ provided in IBM 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 463	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 547	Integrate your applications or client process by using Web services.	Perform when using Web services
"Working with the Java document viewer toolkit" on page 591	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 627	Use the JavaServer Pages (JSP) tag library and a servlet to write JSP or servlets for Web applications.	Perform when necessary
"Troubleshooting content management applications" on page 641	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:

- IBM Information Integrator for Content advanced workflow
- All C++ connectors
- Remote Java connectors
- Image Plus for 390 connector
- VideoCharger player

Important information for Oracle Database 11g users

As of Release 2, Oracle Database 11g is no longer compatible with 32-bit environments on 64-bit operation systems.

Content Manager EE C++ API

If you use Oracle 11g Release 2 and later with the Content Manager EE C++ API on AIX® and Microsoft Windows, you must install and configure the 32-bit Oracle client separately.

Content Manager EE supports installing a 32-bit Oracle client and a 64-bit Oracle server on the same machine, if the client and server each use a different ORACLE_HOME directory. The 32-bit client must be configured as follows, depending on the type of operating system on which it is installed:

Windows

1. Install the 32-bit Oracle client separately from the Oracle server and verify that the OCI Oracle libraries are installed.
2. Set ORACLE_HOME as the home installation directory of the 32-bit Oracle client.
3. Configure the `tnsnames.ora` and `sqlnet.ora` files for the 32-bit client.
4. Add the following text to the beginning of your PATH string: `%32bit_CLIENT_ORACLE_HOME%\bin;`. For example: `PATH= %32bit_CLIENT_ORACLE_HOME%\bin;%PATH%`

AIX

1. Install the 32-bit Oracle client separately from the Oracle server and verify that the OCI Oracle libraries are installed.
2. Add the following text to the beginning of your LIBPATH string: `$32bit_CLIENT_ORACLE_HOME/lib:`. For example: `export LIBPATH=$32bit_CLIENT_ORACLE_HOME/lib:$LIBPATH`

IBM Information Integrator for Content C++ API

Applications that use Oracle 11g Release 2 and later with the IBM Information Integrator for Content C++ API on AIX must be recompiled with version 9 of the xlc/C++ compiler and linked with version 9 of the xlc/C++ archive libraries for IBM Information Integrator for Content (for example, libcmbcm819.a).

However, applications that use Oracle 10g with the IBM Information Integrator for Content C++ API on AIX can be compiled with either version 7 or version 9 of the xlc/C++ compiler. If you use version 7 of the xlc/C++ compiler, it is not necessary to recompile your applications. However if you choose to use version 9 of the xlc/C++ compiler, you must recompile and link your applications with version 9 of the xlc/C++ archive libraries for IBM Information Integrator for Content. DB2® applications can be compiled with either version 7 (no recompile needed) or version 9 of the xlc/C++ compiler, but they must be linked with version 9 of the xlc/C++ archive libraries for IBM Information Integrator for Content.

Deprecated and removed features

The deprecated and removed connector table lists the connectors, APIs, and services that were available in IBM 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.

IBM 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 IBM Information Integrator for Content solution. Additionally IBM Content Manager Version 8.4 deprecates the federated connector (federation), federated advanced workflow, and the OD connector (Content Manager OnDemand).

The following table identifies the status and the changes to the IBM Information Integrator for Content connectors in IBM 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	IBM Content Manager V8	Active
IP	Image Plus (IP/390)	Active
JDBC	Relational connector	Removed
FED	Federation	Deprecated, use Content Integrator
OD	Content Manager 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 IBM 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. IBM Information Integrator for Content supports the following content servers:

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

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

The IBM 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 IBM Information Integrator for Content system administration. IBM 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, IBM 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 IBM 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 run time objects, and therefore do not exist after an application terminates.

"Dynamic data objects (DDO)"

"Extended data objects (XDO)" on page 6

"Representing multimedia content" on page 6

"Content servers and DDOs" on page 7

"Comparing DDOs and XDOs with attribute values and item parts" on page 7

"Persistent identifiers (PID)" on page 7

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. The following figure shows dynamic data objects and data items.

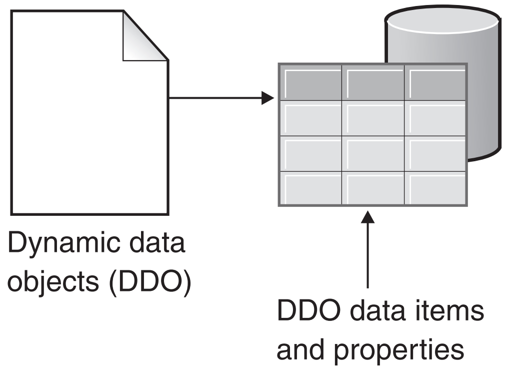


Figure 1. Dynamic data objects and items

Extended data objects (XDO)

An *XDO* is a representation of complex multimedia data, for example a resource item storing an image or document in IBM 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. The following figure shows an example of XDOs.

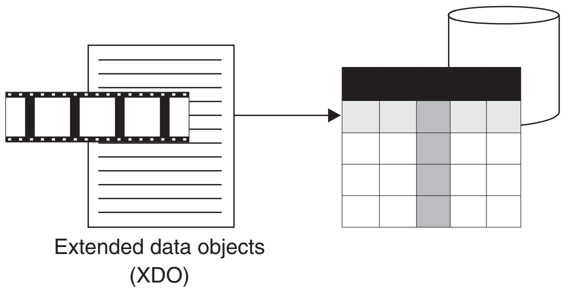


Figure 2. Extended data objects (XDOs)

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 IBM 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 IBM Information Integrator for Content application goes through the following steps 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 DDOs and XDOs with attribute values and item parts

The object type of a DDO corresponds to the associated item type of the item. The data items of a DDO correspond to the attributes of an item.

A DDO corresponds to an item in IBM Information Integrator for Content. For example, in IBM 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 IBM 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.

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 IBM 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 following list describes differences between the IBM 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 the Java garbage collector.
- The Java functions `DKDDO.toXML()` and `DKDDO.fromXML()` are not available in C++.
- The XML classes are available only in Java.

Understanding client/server architecture (Java only)

The APIs provide a convenient programming interface for application writers. APIs can reside on both the IBM 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 Remote Method Invocation (RMI). 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 Version 8.

Packaging for the Java environment

The IBM Information Integrator for Content APIs are contained in four packages.

The four packages that contain IBM 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"

Programming tips

Import the client or the server package depending on the type of application you develop.

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" on page 12
"Setting the Java environment variables for AIX" on page 12

"Setting the Java environment variables for Solaris and Linux"
"Setting the Java environment variables for z/OS USS" on page 13
"Using Remote Method Invocation (RMI) with content servers" on page 13

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 IBM 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 and Linux, use the shell script or set the environment variables.

In the Solaris and Linux environments, you can use a shell script, `cmbenv81.sh`, to set up your development environment for developing IBM 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 IBM 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 compiler flag.

When you set up your Windows or AIX environment, you must establish the settings described in this section. The following table 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 IBM 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 IBM Information Integrator for Content database. For details, see the administering information.

Important: `cmbcm81`*x*`.lib` is for release build and `cmbcm81`*x*`d.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 machine as the Microsoft Visual Studio .NET 2003 compiler. The Microsoft Visual C++ 2005 compiler can also be used to compile library server user exits for IBM Content Manager installed on Windows.

To compile IBM 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 does not compile. For more information, check C++ sample makefiles.

To compile IBM 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 does not compile. For more information, check C++ sample makefiles.

Tip: Some of the IBM Content Manager Version 8 C++ native APIs use the operators typeid and dynamic_cast to access the Run time Type Information (RTTI) of IBM Content Manager objects. This means that your C++ application must be compiled with the `/GR` option to enable RTTI support. See the IBM Content Manager Version 8 C++ sample makefiles for example compiler options.

Table 2. Shared objects and DLL environment information

Connector	Library	Windows DLLs	Shared objects for AIX
Common note: Not a connector. It is a common set of APIs used by all of the connectors.	cmbcm81*x*.lib cmbcm81*x*d.lib	cmbcm81*x*.dll cmbcm81*x*d.dll	libcmbcm817.a
IBM Content Manager Version 8	cmbicm81*x*.lib cmbicm81*x*d.lib	cmbicm81*x*.dll cmbicm81*x*d.dll cmbicmfac81*x*.dll cmbicmfac81*x*d.dll	libcmbicm817.a libcmbicmfac817.so
Federated	cmbfed81*x*.lib cmbfed81*x*d.lib	cmbfed81*x*.dll cmbfed81*x*d.dll cmbfedfac81*x*.dll cmbfedfac81*x*d.dll	libcmbfed817.a libcmbfedfac817.so
Content Manager OnDemand	cmbod81*x*.lib cmbod81*x*d.lib	cmbod81*x*.dll cmbod816*x*d.dll cmbodfac81*x*.dll cmbodfac81*x*d.dll	libcmbod817.a libcmbodfac817.so
ImagePlus for OS/390	cmbip81*x*.lib cmbip81*x*d.lib	cmbip81*x*.dll cmbip81*x*d.dll cmbipfac81*x*.dll cmbipfac81*x*d.dll	Not supported

Table 2. Shared objects and DLL environment information (continued)

Connector	Library	Windows DLLs	Shared objects for AIX
VisualInfo	cmbv481*x*.lib cmbv481*x*d.lib	cmbv481*x*.dll cmbv481*x*d.dll cmbv4fac81*x*.dll cmbv4fac81*x*d.dl	Not supported

"Setting the C++ environment variables for Windows"

"Setting the C++ environment variables for AIX"

"Building C++ programs"

"Setting the console subsystem for code page conversion on Windows" on page 16

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.

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 IBM Content Manager Version 8 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 IBM 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”

“WebSphere Application Server connection pooling for Java API” on page 17

“Native API renovated retrieve options” on page 18

“C++ API scalability and performance: global caching and connection pooling” on page 18

“Using the C++ API to control the timeout of unused connections” on page 21

“Indicating elements fetched when retrieving the list of ACLs” on page 22

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 (that is, 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 ensures 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.

WebSphere Application Server connection pooling for Java API

WebSphere® Application Server connection pooling enables a datastore to scale to an infinite number of users. Connection pooling also improves the performance of your application by reducing logon time and memory usage.

If you want to develop your own custom application and would like to use connection pooling, you must manually configure connection pooling.

The federated connector and IBM Content Manager Version 8 connectors both use the WebSphere Application Server JDBC connection pooling. If the two connectors share the same database, they share the same WebSphere Application Server database connection pool. If you choose different databases when you are installing the two connectors, the connectors use different WebSphere Application Server database connection pools.

Important: When connection pooling is enabled for federated connector or IBM Content Manager Version 8 connectors, the WebSphere Application Server custom applications cannot log on to the IBM Content Manager server.

The steps to configure WebSphere Application Server connection pooling manually when you are designing the custom application are same as the steps to configure WebSphere Application Server connection pooling when you are using the eClient.

Related information

Configuring Websphere Application Server connection pooling for use with the eClient

Native API renovated retrieve options

To improve usability, performance, documentation, and support of the native APIs, IBM 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.

IBM 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 improves 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 those instances that use mid-tier servers such as IBM Document Manager. These environments include custom applications such as Microsoft .NET, COM, and DCOM applications in which 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 IBM Content Manager.
2. Perform an operation (such as run time create, retrieve, update, delete, and query or administration/definition time operations).
3. Disconnect from IBM Content Manager.

The three C++ API features work as follows:

Global cache

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

By enabling a 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, the following types of objects are stored in the global area instead of in 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 configuration file, `*IBMCMROOT*/db2cmv8/cmgmt/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 must access the IBM Content Manager library server can obtain connected and logged-on `DKDatastore` objects from a datastore pool and 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 IBM Content Manager API to pool database connections to the library server database, which 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/cmgmt/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 must 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, ensure that the application closes the pool of connections when it is terminating. The enablement ensures a clean shutdown with a timely release of resources. The following static method in the DKDatastoreICM instance must 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 instead calls clearCache, the DK_CM_CLEAR_CACHE_ALL option 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 the following 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 ensures 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();
```

Using the C++ API to control the timeout of unused connections

IBM Content Manager enables a mechanism for controlling the timeout of unused connections in the database pool.

The C++ API supports a mechanism to pool database connections from IBM Content Manager Version 8 Release 3 Fix Pack 2 and later. The database connection pool is used internally by the IBM 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. Review the type of scenarios under which the C++ API database connection pool must be enabled.

IBM Content Manager enables a timeout mechanism to remove unused connections in the connection pool for the 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 property called `CPPDBConnectionPoolTimeOut` is available to control the timeout of unused connection in the database connection pool.

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

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` property value. If the connection has not been used for this duration, the connection is removed from the pool. Removing unused connections also de-allocates the resources in both the library server and the API. The database timeout mechanism is disabled by setting the value for the `CPPDBConnectionPoolTimeOut` property to 0.

If this value is not set in the `cmbpool.ini` file and the 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.

Related reference

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

Indicating elements fetched when retrieving the list of ACLs

IBM Content Manager provides APIs for the `DKAuthorizationMgmtICM` class that enables applications to indicate the elements that must 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 must be fetched when they list ACLs (* means 0 or more parameters for `listAccessControlLists` and `listUserAccessControlLists`).

IBM Content Manager also provides another API for the `DKAuthorizationMgmtICM` class, `listPrivilegeSets(*, options)` that enables applications to have more flexibility in determining what needs to 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 large results sets on large archives"

Handling large results sets on large archives

If you have large result sets and are primarily interested in retrieving the first page of results as quickly as possible, you can use any of the available methods, such as `execute()`, `evaluate()`, or `executeWithCallback()`. Each method has its advantages.

To retrieve large result sets on large archives, use any of the following methods depending on your requirements:

DKDatastoreICM::execute()

Use `DKDatastoreICM::execute()`, which returns a `dkResultSetCursor`. This method instantiates the DDOs for the first page of results. The smart cursor is optimized, so a retrieve operation is not forced for each fetch request.

The `execute()` method returns an open cursor to the database. When you make the first `fetch()` request, the fetch request retrieves a block of results and creates one block of empty DDOs with completed PIDs. Depending on the retrieve options that you have specified, the `execute()` method calls the multi-item `retrieve` method on that block of DDOs. When you call the `fetchNext()` method again, the function searches for items that are left in a block that is already retrieved and it just returns the next DDO from memory. If you call `fetchNext()` method and there are no items that are left in any pre-fetched block, it loads another block and creates another block of blank DDOs with completed PIDs, and then the `fetchNext()` method sends them to the multi-item `retrieve` method. You can configure the prefetch block size through options, but the default value of the prefetch block size works effectively.

Important: Do not set the block size too large because if you exceed the internal maximum batch size that multi-item retrieve uses, the `execute()` method can be slightly less efficient than when you use the default value of the block size. You should choose block sizes of 300 items or smaller or use the default. The benefits of using the `execute()` method are that if you have 50,000 results, you instantiate and retrieve only 50 – 300 DDOs. The API does not hold the PID information in memory for the other 49,950 items. You can always fetch the first DDO on the next page so that you know if a next page exists. Therefore, if the next page exists, you can display a more button or icon. The disadvantage of using this function is that you have a connection open the whole time.

DKDatastoreICM::evaluate()

The `evaluate()` method instantiates empty DDOs with completed PIDs for all results before returning. You have to wait for the query to read 50,000 PIDs from the library server results, create 50,000 DDOs in memory, and set the PIDs. Depending on your retrieve options, the `evaluate()` method can call multi-item retrieve on the 50,000 DDOs. Using this method is not a good choice for many applications. Retrieving 50,000, 100,000, or 500,000 items even to list query results can affect the efficiency.

You can also use the `evaluate()` method and search your own blocks of results. For example, set the maximum results to a number such as 100 and use SORTBY ITEMID. You get the first 100 results. If you want the next hundred results, run the same query again by using `@ITEMID > insertLastItemIdOfLastBlock` in the predicate so that you get the next 100 results sorted in item ID order. This example is also a good technique for failure recovery even for use of the `execute()` method. If you sortd by ITEMID and know the item ID where you stopped, then if the connection goes down, you could run a query for the remaining results after that item ID. The `TExportManagerICM` sample sorts the results by item ID and it can automatically reconnect and retry or can be restarted later automatically from tracking files.

callback or cancellable callback

Use callback or cancellable callback options with the `DKDatastoreICM::executeWithCallback()` method. This method requests the API to create a separate thread that calls the methods on the object instance of a class that you implement by following a defined interface whenever some results are available. The method calls your object and some more results are passed into the object. You can display results as soon as they are passed into the object and display more as more results are passed to your object over time. However, you will still get the whole 50,000 or 100,000 results and the size of that result set can be a problem if you do not set a limit.

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 in 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 following code example demonstrates the throw and catch process for IBM Information Integrator for Content and IBM Content Manager:

Example: Java

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

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 about 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 3 for a list of the extensions (extensions appended to each `DKDatastore`).

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

Java

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

C++

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

Connecting to content servers

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

If your application terminates abruptly or abnormally, such as when a user enters Ctrl + C, you must ensure that any connected data stores are disconnected by your application. The following table lists each `DKDatastore`*xx* class and its associated content server.

Table 3. Server type and class name terminology

Content server	Class name
IBM Content Manager	`DKDatastoreICM`
Content Manager OnDemand	`DKDatastoreOD`
IBM Content Manager for AS/400 (VisualInfo for AS/400)	`DKDatastoreV4`
ImagePlus for OS/390	`DKDatastoreIP`

“Establishing a connection”
“Connecting and disconnecting from a content server in a client” on page 27
“Setting and getting content server options” on page 27
“Listing content servers” on page 28
“Listing the entities and attributes for a content server” on page 28

Establishing a connection

Each `DKDatastore`*xx* class provides methods for connecting to it and disconnecting from it.

The following example uses an IBM 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("ICMNLSDB","ICMADMIN","PASSWORD","");  //Connect to datastore

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

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

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

C++

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

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

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

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

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

Connecting and disconnecting from a content server in a client

You use the same code 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 DKDatastore*xx*.

The following example shows how to set and get the option for establishing an administrative session on an IBM 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 IBM Content Manager, caching is turned off. For content servers supporting this option, it is recommended to use the default (ON).

Important: In this example, a valid `DKDatastoreICM` object is previously created and is referenced by a variable named `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

`DKDatastore`*xx* provides a method to list the servers that it can connect to.

The list of servers are returned in a `DKSequentialCollection` of `DKServerDef`*xx* objects (where *xx* identifies the specific content server).

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

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

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

`DKDatastore`*xx* objects provide 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 in which a name space is not specified.

The list of entities is returned by the `listEntities` method in a `DKSequentialCollection` object of `dkEntityDef` objects. The attributes for an entity are returned by the `listEntityAttrs` method 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 methods, see the *Application Programming Reference*.

Example: Java

The following example shows how to retrieve the list of item types and the list of item attributes from an IBM 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 an IBM 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 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, for example, an IBM Content Manager document, 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 various content servers and formats. For example, items from different item types in IBM 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"
"Adding properties to a DDO" on page 32
"Creating a persistent identifier (PID)" on page 32
"Working with data items and properties" on page 33
"Getting the DKDDO and attribute properties" on page 35
"Displaying the contents of a DDO" on page 36
"Deleting a DDO (C++ only)" on page 37

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: IBM Content Manager Version 8 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 IBM 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.

"Creating a DKDDO from a PID string"

Creating a DKDDO from a PID string

You can create a DKDDO from a PID string and set the item type property to document or folder.

To create a DKDDO from a PID string:

```
DKDDO createDDO(String pidString);
```

Remember: Before using the DDO that is created for any operations, you must retrieve the DDO by using proper retrieve options.

Related information

Retrieving an item

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, such as the datastore name, datastore type, and object type.

The PID identifies the persistent data location of the DDO. For example, in earlier versions of IBM Content Manager, 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 versions of IBM Content Manager. In IBM Content Manager Version 8 and later, the PID has additional information, such as item ID and component ID.

The IBM Content Manager Version 8 datastore createDDO method creates a PID object for the DDO object created. During the add operation, the datastore populates the PID object with information such as the item ID.

The following examples demonstrate how to create 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();
```

The IBM Content Manager Version 8 connector has a PID class called DKPidICM, which is a subclass of DKPid. The DKPidICM object is used by the DKDDO and 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 Version 8 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 IBM 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: "+attrValr);
```

This example was taken from the `SItemRetrievalICM` sample.

Example: C++

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

DKString attrVal1 = ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,
                    DKString("S_varchar"))).toString();
long     attrVal2 = (long) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,
                    DKString("S_integer")));
short    attrVal3 = (short) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,
                    DKString("S_short")));
DKTimestamp attrVal4 = (DKTimestamp) ddo->getData(ddo->dataId(
                        DK_CM_NAMESPACE_ATTR, DKString("S_time")));

cout << "Attr 'S_varchar' value: " << attrVal1 << endl;
cout << "Attr 'S_integer' value: " << attrVal2 << endl;
cout << "Attr 'S_short'   value: " << attrVal3 << endl;
cout << "Attr 'S_time'    value: " << attrVal4 << endl;
```

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 contents of a 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;). This action 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.

IBM Content Manager Version 8: IBM Content Manager Version 8 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 subclass 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)"

"Java programming tips"

"Programming an XDO as a part of DDO" on page 39

"Programming a stand-alone XDO" on page 40

"Examples of working with an XDO" on page 43

Using an XDO persistent identifier (PID)

An XDO needs a PID to store its data persistently. To use an XDO to locate and store data, you must supply a PID for the DKBlob*xx*, using a DKPidXDO*xx*.

Relational databases require the table, column, and 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 DKPidXDO*xx*.

In the IBM Content Manager Version 8 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.

Java programming tips

For IBM Content Manager Version 8 and later, an XDO is a dkResource object. You use DKPidICM to represent the PID of the resource object.

For earlier versions of IBM Content Manager, IBM Content Manager for AS/400, and ImagePlus for OS/390, you identify an XDO by the combination of item ID, part ID, and RepType. For relational databases, an XDO is identified by a 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 because the system provides a default value for it.

Use the add method of DKBlob*xx* to add the current content to a content server. You can retrieve the part ID value after using add if you want to perform another operation with that object.

Use the getPidObject method of dkXDO to get the DKPid object.

Attention: In earlier IBM 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 is replaced by getPidObject. DKPid dkXDO.getPidObject(). These methods, which previously returned a DKPidXDO object, now 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.

If `DKLobICM` is a part of a parent document, do not use the `update` method directly. The parts can be added or removed by adding or removing them from the parts collection in the document. The parts are modified by modifying them in the document item. Do not call `add`, `update`, or `delete` methods directly on the parts. The document is responsible for managing the parts.

The following example creates a document and adds document parts in IBM 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.

“Deferred read from stream or file from document parts”

Deferred read from stream or file from document parts

Use deferred read to eliminate the holding of large binary content in memory.

Java If you are using the Java Native API, use the DKLobICM::setAddLocation() and setAddLocationOption() methods for a new part or the DKLobICM::setUpdateLocation() and setUpdateLocationOption() methods to update an existing part. If you are not sure which method to set, you can set both setAddLocation() and setUpdateLocation() methods and the correct one is used automatically. See the *Application Programming Reference* for more information on these methods.

C++ If you are using the C++ Native API, you can use the DKLobICM::setInputFilename() method as a deferred read operation for document part content. Submitting an input stream for use on saving is not currently available in C++.

Programming a stand-alone XDO

All of the following examples are specific to IBM Content Manager Version 8.

“Adding an XDO from the buffer” on page 41

“Adding an XDO from a file” on page 41

“Adding an annotation object to an XDO” on page 42

Adding an XDO from the buffer

This example shows how to add an XDO from a buffer in IBM 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 (SResourceItemMimeTyesICM.txt sample)
lob.setMimeType("application/msword");
// Load content into item's local memory
lob.setContentFromClientFile
("SResourceItemICM_Document1.doc");
// Add to datastore with content already in memory
lob.add();
```

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

Example: C++

```
// Create an empty resource object
DKLobICM* lob = (DKLobICM*)
dsICM->createDDO("S_lob", DK_CM_DOCUMENT);
// Set the MIME type (SResourceItemMimeTyesICM.txt sample)
lob->setMimeType("application/msword");
// Load content into item's local memory
lob->setContentFromClientFile("SResourceItemICM_Document1.doc");
("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 IBM 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 (SResourceItemMimeTyesICM.txt sample)
text.setMimeType("text/plain");

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

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

Example: C++

```
// Create an empty resource object
DKTextICM* text = (DKTextICM*) dsICM->createDDO("S_text",
   DK_CM_ITEM);
// Set the MIME type (SResourceItemMimeTyesICM.txt sample)
text->setMimeType("text/plain");

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

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

Adding an annotation object to an XDO

The following example adds an annotation part to a document in IBM 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 SItemTypeCreationICM sample. Assume that you are given an instance of a document already stored persistently in the *ddoDocument* variable. Also, assume that you are given a connected DKDatastoreICM object in variable *dsICM*.

Example: Java

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

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

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

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

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

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

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

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

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

Example: C++

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

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

// Set 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”

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.

IBM Content Manager only: A document is an item that can be stored, retrieved, and exchanged among IBM 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 IBM 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 IBM 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.

The DKFolder attribute is set when the DKParts attribute is set.

Important: IBM 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 contains specific folder functionality provided by IBM 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 can 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 IBM 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 about 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 about creating Folders, refer to the SFolderICM API Education Sample.

In IBM Content Manager Version 8, the folder item does not own the folder contents. An item can 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 IBM 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 be NULL only 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” on page 53

“Using the sequential iterator” on page 53

“Managing memory in collections (C++ only)” on page 54

“Understanding federated collection and iterator” on page 55

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 can use its methods 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 an 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. 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 hold-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;
...                                         // use the attributes
acoll->apply(deleteDKAttributeDef);                // deletes all members
delete acoll;
```

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

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

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

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

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.

The following figure shows the structure and behavior of `DKFederatedCollection`.

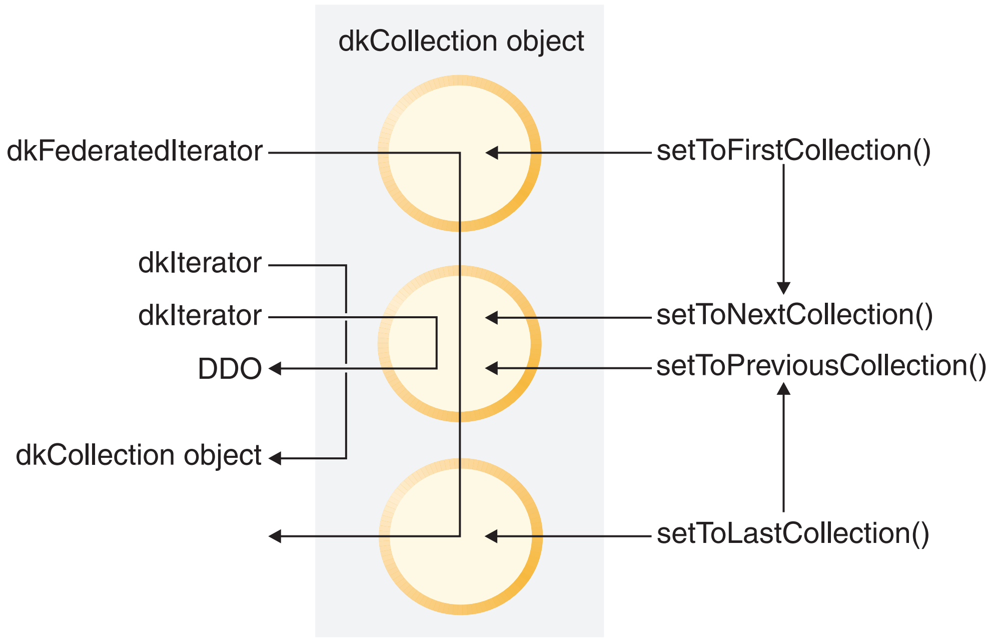


Figure 3. Structure and behavior of DKFederatedCollection

In the previous figure, 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 IBM 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"

"Using parametric queries"

Differences between dkResultSetCursor and DKResults

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

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

- The `dkResultSetCursor` collection functions as a database cursor. It can be used for large result sets because the DKDDO objects 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` collection contains the entire result set and supports a bi-directional iterator.
- Leaving a `dkResultSetCursor` collection open for long periods of time might 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.

Related information

Querying a content server

Working with IBM Content Manager Version 8.4

This section describes the IBM Content Manager Version 8.4 connector (ICM connector) application programming interfaces (APIs). Because the ICM connector is an extension of the IBM Information Integrator for Content framework, you must understand the IBM Information Integrator for Content framework.

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

Important: The ICM connector does not support RMI configuration.

Understanding the IBM Content Manager system

The main components of the IBM 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 IBM 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 IBM 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 database managed storage (DMS).
- 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 IBM Content Manager system. The APIs are available for Java and C++. Using the APIs, your applications can take advantage of all of the IBM Content Manager functionality, such as data modeling, integrated parametric and text search, third-party data access and delivery, and so on.

The following diagram illustrates how the system components fit together. Keep in mind that this diagram shows only one implementation of an IBM Content Manager system. For example, in another system configuration, you might have four resource managers.

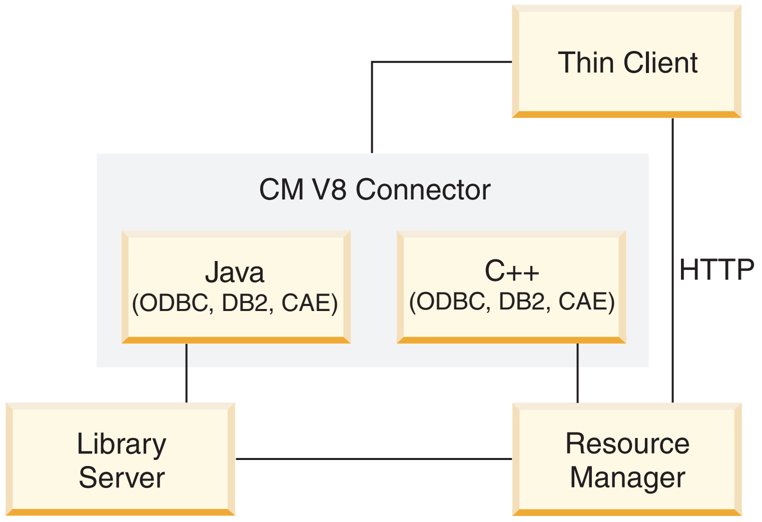


Figure 4. System configuration

Understanding IBM Content Manager concepts

This section describes important IBM Content Manager concepts.

It is imperative that you understand the IBM Content Manager concepts before you proceed to the programming tasks.

"Items" on page 61
"Attributes" on page 61
"Item types" on page 61
"Root and child components" on page 62
"Objects" on page 63
"Links and references" on page 63
"Enabling auto-linking with version-enabled item types" on page 64
"Documents" on page 65
"Folders" on page 65
"Versioning" on page 66

"Access control" on page 66

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, IBM 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 IBM 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 IBM Content Manager versions) is essentially a template for defining and 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 such as policy number, name, claim, and so on.

In IBM Content Manager, there are the following classifications of item types:

- Non-resource: represents entities that are not stored in a resource manager
- Resource: represents objects stored in a resource manager, such as files in a file system, video clips in a video server, LOBs (large objects) in database tables, and so on. IBM Content Manager provides a base set of resource item types: LOB, text, image, stream, and video objects.

- Document (also known as document model): represents entities that contain document parts, each of which contains resource content that is the same type as a single resource item type
- Document part: represents objects that are stored in the resource manager, but are parts of a document, contained and owned by a document item type.
- Hierarchical: represents objects that are stored in a rooted hierarchical folder structure.

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 an l 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 an item type. Each child component is directly associated with the level above it. The following figure shows the diagram of the IBM Content Manager meta-model. It shows root and child components and their relationships. The diagram also shows links, references, resource items, and resource objects.

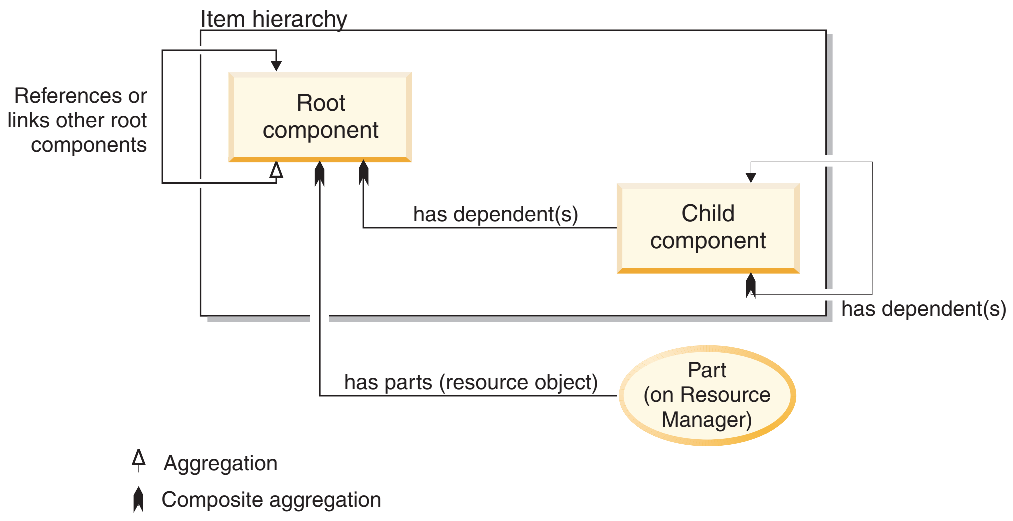


Figure 5. The IBM Content Manager meta-model: A logical view.

The following figure shows an example of a data model for an insurance application. The policy is the root component and the claim, police report, and damage estimate are the child components.

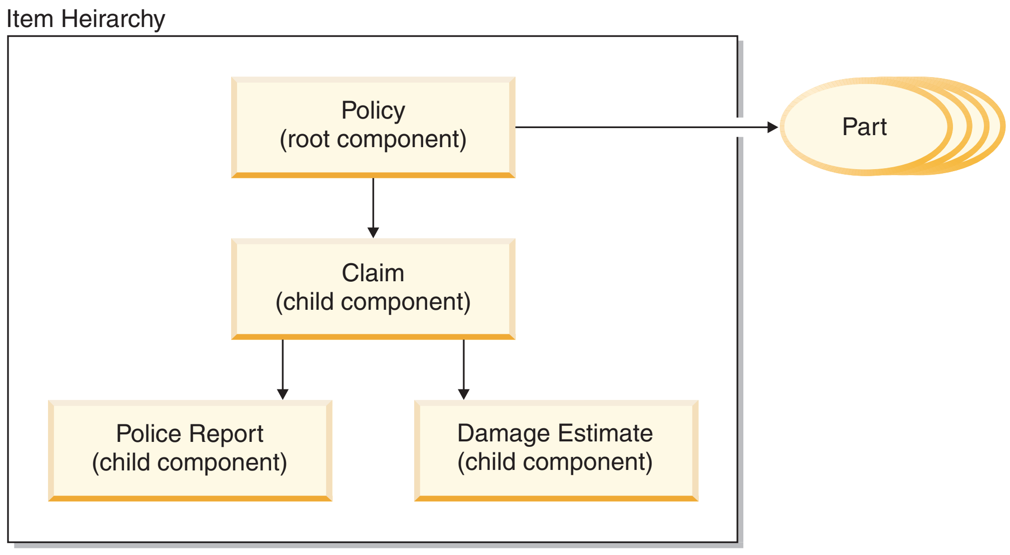


Figure 6. Components hierarchy

For more information about IBM 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 5 on page 62, links can be used only 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 run time by the user application. You can use any number of links in your application.

Links are open and non-restrictive, flexible building blocks that you can use in your applications. As such, you have the option of placing any further restrictions on links within your application.

One of the ways that you can use a link is to represent the foldering or container-containee relationship. If you choose to implement foldering using links, remember that the container does not own the containee, which means that the items in the container are not deleted when the container is deleted. For more information about links, see the `SLinksICM` sample.

A *reference* is between a component (either root or child) and another root component. A reference is represented as a reference attribute in a component defined at design-time. A component definition can have any number, specified during component type definition, of reference attributes that refer to other root components. A reference usually does not indicate ownership, but you can implement an ownership relationship if necessary.

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

References can be defined in both root and child component types referring to another root component type, see Figure 5 on page 62. 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

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

The earlier versions of IBM Content Manager provide a more restricted implementation of auto-linking, which was called auto-foldering. Auto-foldering was restricted to linking folders only. However, IBM Content Manager enables auto-linking with version-enabled item types since Version 8.3 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 V8.3 Fix Pack 4, IBM 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, IBM Content Manager applies the following rules:

Update to the latest version of target item
If the update is for a linking attribute, IBM 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, IBM 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, IBM 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 IBM 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 IBM 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 creates a new version only when specified by the user application.

Versioning is handled by the IBM 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 IBM 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 IBM 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 IBM Content Manager is applied to the controlled entity. A *controlled entity* is a unit of protected user data. In IBM 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. The following figure shows an example of how the system determines user's access rights to an item based on privileges and ACLs.

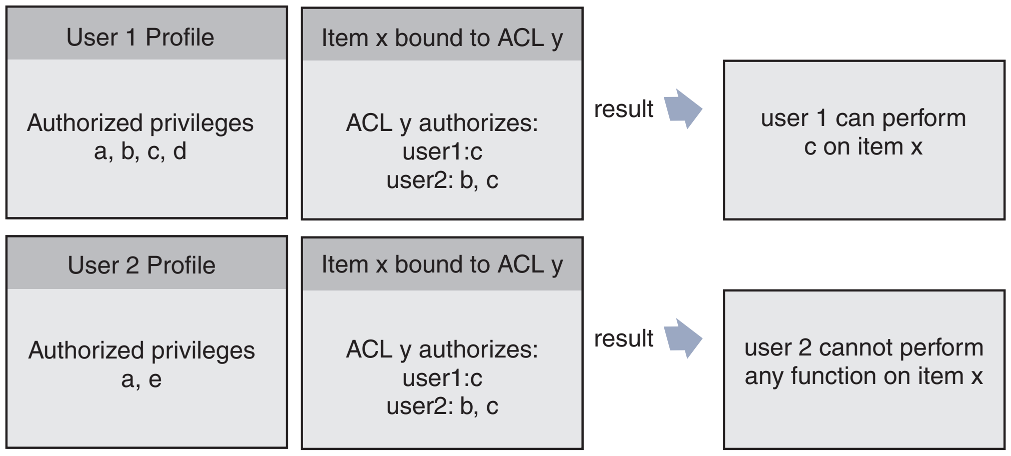

User 1 Profile
Authorized privileges a, b, c, d
Item x bound to ACL y
ACL y authorizes: user1:c user2: b, c
result
user 1 can perform c on item x
User 2 Profile
Authorized privileges a, e
Item x bound to ACL y
ACL y authorizes: user1:c user2: b, c
result
user 2 cannot perform any function on item x

Figure 7. Access control diagram

"Privileges and privilege sets"

"Users and user groups" on page 69

"Access control lists (ACLs) and ACL binding level rules" on page 70

"ACL configuration for document model" on page 73

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 IBM Content Manager user is granted a set of user privileges. The privileges define the maximum operations a user can perform on information in the IBM Content Manager system. A user's access rights do not exceed the defined user privileges for the user.

IBM 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 IBM 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 IBM 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 IBM 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 IBM 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 IBM Content Manager entities. The privileges contained in this privilege set include: None.

SystemAdminPrivSet; PrivSetCode: 3
Users with this privilege set can perform all IBM 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 IBM 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 4. 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 IBM 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 a 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 (ICMPUBLC), 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 ICMPUBLC, 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 ICMPUBLC
- 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 run time 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 an IBM Content Manager application

This section helps you identify requirements for creating an IBM Content Manager application and provides information about how IBM 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 IBM Content Manager data model, see the `tSItemTypeCreationICM` sample in the `samples` directory.

"Determining the features of your application"

"Handling errors"

Related information

Defining data model options

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.

Related information

Planning and installing your content management system

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. The following table contains `DKException` information.

Table 5. DKException information

DKException	Description
Name	Exception Class Name. Contains subclass 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.

Related information

Logging and tracing

Enabling IPv6 dual-stack support

IBM Content Manager 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: IBM Content Manager provides IPv6 dual-stack support for ICM connectors only for both 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/icmnlsdb
  ```

  .

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 6. Preferences to use IPv4 or IPv6

java.net.preferIPv4Stack	**java.net.preferIPv6Addresses**	**default address**
true	true	IPv4
true	false	IPv4
false	false	IPv4
false	true	IPv6

If your TCP/IP is configured as dual-stack to support both IPv6 and IPv4, and the host name for the resource manager specified in the ICMSTResourceMgr table, then the TCP/IP service will resolve the host name to an IPv6 address. In this situation, you must configure the resource manager to use IPv6 by using an Apache server as a reverse proxy. You must also specify that the port column in the ICMSTResourceMgr table correspond to the IPv6 port of the Apache Server reverse proxy (rather than the IPv4 port of the resource manager). Otherwise, the connection from ICMFETCHCONT UDF to the resource manager might fail.

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

IBM Content Manager 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 affects the ICM connector and the federated connector.

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

As of Version 8.4, IBM 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 IBM 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 IBM 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 IBM Content Manager Version 8.4.1, 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.

Important: IBM Content Manager requires a dynamic library (`db2jcct2.dll` on Windows® or `libdb2jcct2.so` on UNIX®) to use a JDBC type 2 driver.

Windows
Add the path that includes the `db2jcct2.dll` file to the PATH variable. By default, the `db2jcct2.dll` file is located in `%DB2HOME%\bin.` directory.

UNIX IBM Content Manager load the `libdb2jcct2.so` file from DB2 instance library path defined in LD_LIBRARY_PATH or LIBPATH variable.

32-bit DB2 instance
DB2 instance library path must exist in LD_LIBRARY_PATH (Solaris® or Linux) or LIBPATH (AIX®).

Solaris
```
export LD_LIBRARY_PATH=/export/home/db2inst1/sqllib
/lib:$LD_LIBRARY_PATH
```

Linux
```
 export LD_LIBRARY_PATH=/home/db2inst1/sqllib/lib:$LD_LIBRARY_PATH
```

AIX
```
export LIBPATH=/home/db2inst1/sqllib/lib:$LIBPATH
```

64-bit DB2 instance

For 64-bit DB2 instance, `[DB2 Instance Dir]/sqllib/lib` links to 64-bit library directory. The link relationship of the DB2 library directory on UNIX is as follows:

```
[DB2 Instance Dir]/sqllib/lib ->
[DB2 Instance Dir]/sqllib/lib64 ->
[DB2 Product Installation Dir]/lib64
[DB2 Instance Dir]/sqllib/lib32 -> [DB2 Product Installation Dir]/lib32
```

But, IBM Content Manager supports only the 32-bit `libdb2jcct2.so` since Version 8.4. To make IBM Content Manager and its APIs work correctly, the path to 32-bit libraries must be listed before the path to the 64-bit libraries in LD_LIBRARY_PATH or LIBPATH. For example, on AIX:

```
export LIBPATH=/opt/IBM/db2/V9.1/lib32:$LIBPATH
```

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 IBM 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 IBM Content Manager.

Important: The file (`db2java.zip`) is no longer used and can be removed from the class path of any scripts.

Modifications to cmbicmsrvs.ini and cmbds.ini files

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

IBM Content Manager also allows you 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, IBM Content Manager 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 `JDBCDRIVER` properties by using the **`connect_string`** parameter associated with the `DKDatastoreICM::connect` method.

In IBM 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 `com.ibm.db2.jcc.DB2Driver` class.
- If the `ICMJDBCURL` property is specified and the ICMHOST, ICMPORT, and ICMREMOTEDB information is also specified in the `cmbicmsrvs.ini` and `cmbds.ini` files, 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 `cmbicmsrvs.ini` file and in the `cmbds.ini` file, the `JDBCURL` property in the **`connect_string`** parameter takes precedence. Similarly, the `JDBCDRIVER` property

passed through the **connect_string** parameter takes precedence over the ICMJDBCDRIVER property specified in cmbicmsrvs.ini file and in the cmbds.ini file.

Setting both the URL and the driver is not a requirement since IBM 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. Since Version 8.4, IBM Content Manager supports only the IBM DB2 Driver for JDBC and SQLJ when connecting to a DB2 database. IBM 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/icmnlsdb
```

Remember:

When an IP address is used instead of a host name in the JDBC URL, square brackets ([and]) must enclose the IP address if IPv6 is used. For example, the connect string for IPv6 address 2007::9:181:141:150 must be entered in the following format:

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

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

Related information

Configuring the database connection

Supported drivers for JDBC and SQLJ

JDBC connection string support in Oracle

Content Manager EE supports the Oracle JDBC driver (type 2 and type 4), which affects both the ICM connector and the federated connector. Content Manager EE also supports flexible JDBC connection strings, which allow you to use a single and consistent connection string across components.

If you are using Content Manager EE with Oracle 11g, you must you use a JDBC type 4 driver; type 2 drivers are not supported. The type 4 driver is a pure Java™ driver. For this type of driver, the application requires only the Oracle JAR files, not the full Oracle client. However, you must specify the ICMJDBCURL field in the cmbicmsrvs.ini file for the ICM connector and in the cmbds.ini file for the federated connector. If the ICMJDBCURL field is not specified, a JDBC type 2 connection string is used by default. You must ensure that Oracle Net is configured on your Oracle server and client for the connection string type that you want to use. For more information about JDBC connections strings, see *Oracle Database JDBC Developers Guide and Reference*.

JDBC connection strings

Content Manager EE provides flexibility in working with JDBC connection strings by using the existing ICMJDBCURL field in the `cmbicmsrvs.ini` file. Content Manager EE allows you to specify your connection string. When you install Content Manager EE, the JDBC connection string that you provide is used to configure the APIs.

If you do not want to use the default JDBC type 2 connection string, you must manually update the ICMJDBCURL field in the `cmbicmsrvs.ini` file with your connection string.

C++ If you use the IBM Information Integrator for Content C++ connectors to connect to a Content Manager EE library server on Oracle, the ICMSERVER value for the library server in the `cmbicmsrvs.ini` file must match the ORACLE_SID value corresponding to your library server database. These values must match, because the C++ APIs use the ICMSERVER value when calling Oracle's `OCIServerAttach()` function to establish a connection to the library server database. When you install Content Manager EE, the ICMSERVER value is set to the default value, which is the library server name that you selected during installation. If you want to use the C++ APIs, set the ICMSERVER value in the `cmbicmsrvs.ini` file to the ORACLE_SID value. If you upgrade to the latest version of Content Manager EE from previous versions, it is not necessary to change the `cmbicmsrvs.ini` file because Content Manager EE does not modify the `cmbicmsrvs.ini` file.

Content Manager EE supports the following Oracle connection string types:

- Oracle Call Interface (OCI) type 2 connection: for example, jdbc:oracle:oci:@myhost.mydomain.com:1521:icmlsdb. OCI type 2 connections are not supported by Content Manager EE if you are using Oracle 11g
- OCI type 4 (thin) connection: for example, jdbc:oracle:thin:@//myhost.mydomain.com:1521/icmnlsdb
- Thin
- Thin-style service name
- TNSNames alias
- Bequeath connection
- LDAP syntax
- Oracle Net connection descriptor

Important: If the ICMJDBCURL field is not specified in the `cmbicmsrvs.ini` file, the API uses a JDBC type 2 connection string, which is based on the library server name. If you want to use a JDBC type 4 connection string, you must specify a value for ICMJDBCURL field in the `cmbicmsrvs.ini` file by reinstalling or by manually updating the `cmbicmsrvs.ini` file.

Modifications to cmbicmsrvs.ini and cmbds.ini files

Content Manager EE provides the `ICMJDBCURL` and `ICMJDBCDRIVER` properties in the `cmbicmsrvs.ini` file for the ICM connector and in the `cmbds.ini` file for the federated connector. You can also provide the `JDBCURL` and `JDBCDRIVER` properties by using the **`connect_string`** parameter that is associated with the `DKDatastoreICM::connect` method.

Important: For Oracle, if you install Content Manager EE, the value of the `JDBCURL` property is stored as `ICMJDBCURL` in the `cmbicmsrvs.ini` file for the ICM connector and in the `cmbds.ini` file for the federated connector. `ICMJDBCURL` is used by the `connect` method and the default JDBC type 4 connection string. If you upgrade from a version prior to 8.4.1, `ICMJDBCURL` does not exist and therefore the JDBC type 2 connection string is used.

In Content Manager EE, these properties are optional and do not have any default values. The values of these properties are specified as follows:
- If the `ICMJDBCDRIVER` property is not specified or is empty, the Oracle JDBC driver is used by default by using the oracle.jdbc.driver.OracleDriver class.
- 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 `JDBCDRIVER` property that is passed through the **`connect_string`** parameter takes precedence over the `ICMJDBCDRIVER` property that is specified in the `cmbicmsrvs.ini` file and in the `cmbds.ini` file.

Related information

Connection to an Oracle library server database fails

Working with the IBM Content Manager samples

IBM Content Manager provides a comprehensive set of code samples to help you complete key IBM 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 IBM Information Integrator for Content installation in order to have the samples in the directory.

Remember: The IBM 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”

The insurance scenario sample

IBM 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 IBM Content Manager features. For a complete list of the samples that make up the insurance scenario, see the Samples Readme.

Creating an IBM Content Manager application

The APIs that implement IBM 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”

“Representing items using DDOs” on page 83

“Connecting to the IBM Content Manager system” on page 83

“Working with items” on page 84

“Working with hierarchical item types” on page 117

“Working with folders” on page 139

“Defining links between items” on page 147

Understanding the software components

You can categorize the 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 IBM Content Manager data model. For example, the data model of an insurance company includes policies, which in the IBM 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 IBM 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 IBM 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 on.

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 IBM Content Manager.

In memory, IBM 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 IBM Content Manager system

One of the first things that you must do when you build an IBM Content Manager application is connect to the server.

This section helps you with the various tasks involved in connecting to, and disconnecting from, an IBM 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 IBM Content Manager server, your application must create a content server, which acts as a common server. To create and connect to a content server:

1. Create a content server object.

   Java
   ```
   DKDatastoreICM dsICM =new DKDatastoreICM();
   ```
   C++
   ```
   DKDatastoreICM *dsICM =new DKDatastoreICM();
   ```
2. Set up the connection parameters.

   Java
   ```
   String database = "icmnlsdb";
   String userName = "icmadmin";
   String password = "password";
   ```
   C++
   ```
   char * database = "icmnlsdb";
   char * userName = "icmadmin";
   char * password = "password";
   ```
3. Call the `connect` operation on the content server. `databaseNameStr` is the name of the database you want to connect to.

   Java
   ```
   dsICM.connect(databaseNameStr,usridStr,pwStr,"");
   ```
   C++
   ```
   dsICM->connect(database, userName, password, "");
   ```

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 previous code snippets were extracted.

"Changing a password"

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" on page 85

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. IBM Content Manager supplies the following item type classifications: item, resource item, docmodel, and docpart. If you do not specify an item type classification when you create an item type, the item type classification defaults to item (DDO). The following table contains the pre-defined item type classifications and their corresponding ID constant.

Table 7. Item type classifications

Classification	ID Constant	Number	Description
Item	`DK_ICM_ITEMTYPE_CLASS_ITEM`	0	A standard item (DDO).

Table 7. Item type classifications (continued)

Classification	ID Constant	Number	Description
Resource	DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM	1	A resource item that describes and contains data that is stored in the resource manager. Video, images, documents, and other data archived in the resource manager are examples of resource items.
Document model	DK_ICM_ITEMTYPE_CLASS_DOC_MODEL	2	An item that models documents using parts. A document is composed of any number of parts, which are contained in the attribute DKConstant.DK_CM_DKPARTS.
Part	DK_ICM_ITEMTYPE_CLASS_DOC_ PART	3	Items (parts) in the document model classification.
Notes: 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.addAttr(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 itemTypeName[] = dsICM.listEntityNames();
DKSequentialCollection itemTypeColl =
  (DKSequentialCollection) dsICM.listEntities();
dkIterator iter = itemTypeColl.createIterator();
while(iter.more()){
dkEntityDef itemType = (dkEntityDef) iter.next();
System.out.println(" Item type name : " + itemType.getName());
}
```

C++ Example 1

```
long larraySize = 0;
DKString * itemTypeName = dsICM->listEntityNames(larraySize);
   for (int i = 0; i < larraySize; i++) {
           cout <<(char*)itemTypeName[i] <<endl;
    }
   delete [] itemTypeName;
```

C++ Example 2

```
DKSequentialCollection * itemTypeColl =
                            (DKSequentialCollection *)
            dsICM->listEntities();
```

```
dkIterator * iter = itemTypeColl->createIterator();
  while (iter->more()) {
dkEntityDef* itemType=(dkEntityDef*)((void*)(*iter->next()));
   cout <<(char*)itemType->getName() < delete(itemType);
  }
  delete(iter);
  delete(itemTypeColl);
```

For more information, see the `SItemTypeRetrievalICM` sample.

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 IBM 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 on. Attribute names are limited to 32 characters. If you must include more information, use the description field. The following table contains examples of types of attributes that you can create:

Table 8. Attribute types

Type	Constant	Object
BLOB	`DKConstant.DK_CM_BLOB`	byte bytes[]
Char	`DKConstant.DK_CM_CHAR`	java.lang.String
CLOB	`DKConstant.DK_CM_CLOB`	java.lang.String
Date	`DKConstant.DK_CM_DATE`	java.sql.Date
Decimal	`DKConstant.DK_CM_DECIMAL`	java.math.BigDecimal
Double	`DKConstant.DK_CM_DOUBLE`	java.lang.Double
Integer	`DKConstant.DK_CM_INTEGER`	java.lang.Integer
Short	`DKConstant.DK_CM_SHORT`	java.lang.Short
Time	`DKConstant.DK_CM_TIME`	java.sql.Time
Timestamp	`DKConstant.DK_CM_TIMESTAMP`	java.sql.Timestamp
Varchar	`DKConstant.DK_CM_VARCHAR`	java.lang.String

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

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.

Adding the ICM$NAME attribute to a nonhierarchical item type

The ICM$NAME attribute is the standard name attribute for an item type. It is an optional attribute for nonhierarchical item types and is required for hierarchical item types.

You can add the ICM$NAME attribute to the root component of any new or existing item type, except the part type class. The ICM$NAME attribute can be applied to documents, folders, or any data that is returned while performing a browse or a search. As the standard name attribute, the ICM$NAME attribute is useful for sorting and querying operations on item types, such as sorting results by name and resolving name-based paths.

The ICM$NAME attribute has the following characteristics:
- It is a predefined user attribute.
- It cannot be updated or deleted.
- Its value does not have trailing blanks
- Its data type is VARCHAR with length 256.
- It cannot be added to any user-defined attribute groups.
- When it is created, its attribute ID conforms to the user attribute ID generation rule (1 is added to the maximum). Therefore, its attribute ID is greater than or equal to 1000 and is different on different Content Manager EE systems.

When the ICM$NAME attribute is added to a nonhierarchical item type, it has the following default component attribute properties. Unless otherwise noted, these properties can be changed:
- Nullable, if added to an existing nonhierarchical item type. This property cannot be changed.
- Not nullable, if added during the creation of a nonhierarchical item type
- Not unique. This property cannot be changed.
- Not text-searchable
- Not represented
- No default value

When you provide a value for ICM$NAME, there is no restriction on the characters you can use. However, all the trailing blanks in the value are removed by the library server. Therefore, if you have provided an ICM$NAME value with trailing blanks, those trailing blanks are not visible in any search results. If you provide an ICM$NAME value with a string consisting entirely of blanks, its value is converted to an empty string and error ICM7016 is returned from the library server.

The following restrictions apply for an ICM$NAME attribute of a nonhierarchical item type. You cannot:

- Update the uniqueness property of the ICM$NAME attribute after it has been added to an item type. If you attempt to do so, the library server returns error ICM7123.
- Add the ICM$NAME attribute to a child component. If you attempt to do so, error DGL7351A (`DK_ICM_MSG_HIERARCHICAL_CHILD_COMPONENT_USE_ICMNAME_INVALID`) is returned.
- Add the ICM$NAME attribute to a user-defined attribute group. If you attempt to do so, error DGL7353A (`DK_ICM_MSG_HIERARCHICAL_ATTR_GROUP_USE_ICMNAME_INVALID`) is returned.

To get the ICM$NAME attribute value of an item type, call the `getName()` method of its `DKDatastoreICM` object. For more information, see the *Application Programming Reference*.

Related information

Adding the ICM$NAME attribute to a hierarchical item type

ICM$NAME attribute

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 attribute
//definitions 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 `DK_CM_PROPERTY_ITEM_TYPE` property. Do not confuse this property with the overall item type definition that describes 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.

The following table shows a list of available item property types.

Table 9. Item type property definitions

Property type	Constant	Definition
Document	`DK_CM_DOCUMENT`	Item represents a document or stored data. The information contained in this item might form a document. This item can be considered a common document because it does not rigidly conform to an implementation of a specific document model. Setting a document type does not necessarily mean a document with parts using the IBM Content Manager Version 8 document model, which is instead controlled by the "document" classification in the item type definition.
Folder	`DK_CM_FOLDER`	Item represents a folder that can contain other items. This item features a built-in DKFolder collection, which can hold folder contents that form folder links to other items. An empty DKFolder collection is automatically added. All items with this value can use the DKFolder mechanism. However, this value does not necessarily mean that a solution uses the DKFolder mechanism; it can instead implement its own folder solution. This value means any interpretation of a folder, except the built-in DKFolder, is available to such items. **Tip:** You can create non-folder links and references from items of any semantic type or item property type.
Item (default)	`DK_CM_ITEM`	Generic item. This item is the item property type for all semantic types that are not document or folder.

Items are created as DKDDOs. Always use the `createDDO()` methods of the `DKDatastoreICM` object to create `DKDDO` objects because the system uses the `createDDO` of the `DKDatastoreICM` object to automatically set up important information in the DKDDO structure. DDO Constructors are not recommended. Using the various `DKDatastoreICM::createDDO()` methods are the 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 subclass 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 `createDDO` function of the data store. In general, a semantic type 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 IBM Content Manager server. Semantic types give items an enforceable characteristic for some basic operations and provide applications with the ability to apply a semantic type. Where applicable, the semantic types are enforced within IBM 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 object type of the PID, which identifies the exact item type or you can retrieve the definition to tell you more, such as classification as a part, XDO class, and so on. 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 `DKConstantICM.DK_ICM_PROPERTY_SEMANTIC_TYPE` property, which might contain the same value as the item property type. The following table lists the available semantic types.

Table 10. Pre-defined semantic types

Semantic type	Constant	Definition
Any	User-specified	`CreateDDO` assigns 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 IBM 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.

Table 10. Pre-defined semantic types (continued)

Semantic type	Constant	Definition
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 that form folder links to other items. An empty DKFolder collection is automatically added. All items with this value can use 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, except the built-in DKFolder, is available to such items. 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.
Base	`DK_ICM_SEMANTIC_TYPE _BASE`	Most nearly resembles 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 document part a BASE part. This value is not validated or enforced. The item type of the part (identified in the object type of the PID 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 object type of the PID 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) and most nearly resembles "BASETEXT".
BaseStream	`DK_ICM_SEMANTIC_TYPE_BASESTREAM`	Same as BASE (See "Base" documentation for details) and most nearly resembles "BASETSTREAM".
Annotation	`DK_ICM_SEMANTIC_TYPE _ANNOTATION`	Same as BASE (See "Base" documentation for details) and most nearly resembles "ANNOTATION".
History	`DK_ICM_SEMANTIC_TYPE _HISTORY`	Same as BASE (See "Base" documentation for details) and most nearly resembles "HISTORY".
Note	`DK_ICM_SEMANTIC_TYPE _NOTE`	Same as BASE (See "Base" documentation for details) and most nearly resembles "NOTE".

Table 10. Pre-defined semantic types (continued)

Semantic type	Constant	Definition
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 also create any child components that make up the item, if they exist.

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 IBM Content Manager Version 8, you can pre-define foreign key attribute values to populate client drop-down menus with specific choices.

These values can be associated with an attribute of an item type. For example, an auto claims item type can have a city attribute with the drop-down values of New York and Chicago. Similarly, a life claim item type can have a city attribute with drop-down values of Boston, Atlanta, and Dallas.

IBM 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 the City drop-down list can be populated with San Jose and Los Angeles. To set this relationship, you can define a foreign key item type called StateCity with two attributes: State and City. You can 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 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 value to FALSE, then a text entry field is displayed instead.

The following examples use `getSpecificAttrsForForeignKey()` to return a set of results based on a search of provided attribute (column) names in foreign key and their corresponding values. This method can be used only with foreign keys related to IBM 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 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 are 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 the following table.

Table 11. Example results for parameters passed into getSpecificAttrsForForeignKey()

DKNVPairs	Attribute names	Result
DKNVPair fKeyValues[0](“Country”, “US”);	columnNames[0] = “city”	Returns the names of all cities in the U.S.
DKNVPair fKeyValues[0](“Country”, “US”) DKNVPair fKeyValues[1](“State”, “California”)	columnNames[0] = “City” columnNames[1] = “Zip”	Returns the names of all cities (and their zip codes) in the state of California only
DKNVPair fKeyValues[0](“Country”, “US”) DKNVPair fKeyValues[1](“State”, “California”) DKNVPair fKeyValues[2](“City”, “San Jose”)	columnNames = null;	Given that the foreign key contains the attributes: Country, State, City, addresses, zip, and phone number, then returns all values for address, zip, and phone number.

The following examples return foreign key string arrays in the following order: column sequence number, source attribute, target attribute, and display flag value.

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 must be updated in persistent storage to make the new value permanent. This method can be used only with foreign keys related to IBM 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 can be displayed 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.

Modifying an item's attributes

To modify an item's attributes, use the DDO's `setData` method.

Exception: You cannot modify any of the pre-existing attributes (for example, changing the length of them).

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 your are done updating it. For more information on updating items, see the `SItemUpdateICM` API education sample.

Related information

Retrieve a document's DKLobICM objects in the DKParts collection before you make an update() call

Defining a resource item type

A *resource item* is an item with additional system-defined attributes. These attributes define features, such as the location, type, and size, of the object that the item represents.

If a resource item is enabled for version support and you want to change its attributes in a new version, you must create a copy of its content when you update its attributes.

The object that a resource item represents is called a resource and can be a video file, an image, a word processor document, and so on. For additional information, see the `SItemTypeCreationICM` sample, located in the `samples` directory.

The following steps demonstrate 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 an item type definition.

   Java
   ```
   DKItemTypeDefICM  itemType = new DKItemTypeDefICM(dsICM);
   itemType.setName("SampleResource");
   itemType.setDescription("Simple Resource Lob Item Type");
   ```
   C++
   ```
   DKItemTypeDefICM* itemType = new DKItemTypeDefICM(dsICM);
   itemType->setName("SampleResource");
   itemType->setDescription("Simple Resource Lob Item Type");
   ```
3. Add an attribute.

   Java
   ```
   DKAttrDefICM attr =(DKAttrDefICM)dsDefICM.retrieveAttr("S_varchar");
   itemType.addAttr(attr);
   //Resource classification indicates that this class will
   //contain data file
   itemType.setClassification
     (DKConstantICM.DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
   ```
   C++
   ```
   DKAttrDefICM * attr = (DKAttrDefICM*) dsDefICM->
   retrieveAttr("S_varchar");
   itemType->addAttr(attr);
   // Resource classification indicates that this class will contain
   //data file
   itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
   ```
4. Specify the XDO class and type of resource for this item type.

   Java
   ```
   itemType.setXDOClassName(DKConstantICM.DK_ICM_XDO_LOB_CLASS_NAME);
    itemType.setXDOClassID(DKConstantICM.DK_ICM_XDO_LOB_CLASS_ID);
   itemType.add();
   ```
   C++
   ```
   itemType->setXDOClassName(DK_ICM_XDO_LOB_CLASS_NAME);
   itemType->setXDOClassID(DK_ICM_XDO_LOB_CLASS_ID);
   itemType->add();
   ```

Creating a resource item

Creating resource items is similar to creating regular items. An XDO extends a DDO, and depending on the type of resource item, the XDO can be extended further.

The following table 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 12. 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 following steps 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 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 subclass (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 Microsoft Word document. The MIME type describes the type of content that is being stored.

   Java
   ```
   lob.setContentFromClientFile(fileName);
   lob.setMimeType("application/msword");
   ```

   C++
   ```
   lob->setContentFromClientFile(fileName);
   lob->setMimeType("application/msword");
   ```

   For additional information about MIME types, see the `SResourceItemMimeTypesICM` sample.

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

   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 of the item, before it was stored in IBM 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 the retrieve method (single-item or multi-item) and request options that at least include the system resource attributes, such as DKRetrieveOptionsICM::baseAttributes(true). You can optionally add a filter to retrieve 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).

If a resource item is enabled for version support and you want to change its attributes in a new version, you must create a copy of its content when you update its attributes.

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 IBM 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

IBM 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*.

IBM 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.

Retrieve PID validation or unspecified version number 0

For Java APIs, the basic validation for PID completion is relaxed to allow unspecified version number 0 without requiring you to also specify a retrieve option for the latest version. Individual DDOs with unspecified version 0 in their PID now retrieve the latest version for that particular DDO, regardless of whether you specify the retrieve option for the latest version. You can request the latest version for individual DDOs among collections submitted to multi-item `retrieve` method. You can submit DDOs from retrieved links and folder contents listings without specifying the retrieve option for the latest version.

"Retrieving an item: Java"

"Retrieving an item: C++" on page 109

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 re-create 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 `createDDOFromPID(DKPidICM pid)` or `createDDOFromPID(String pidString)` method of the `DKDatastoreICM` object. Do not use the DKDDO constructor. You can use this method to re-create 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 multiple users from changing 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 referenced only, not owned. The persistent write-lock on the item is released when you check in the item.

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

A write-lock is persistent and tied to your user name, not your connected session. A write-lock remains in effect until either you or a user with the ItemSuperCheckIn privilege unlocks your item. Furthermore, the same user is allowed to make changes in any authenticated session. If you connect using the same user name to multiple connections, each connection has permission to update the write-lock.

You can retrieve the checked-out status, which indicates the current lock owner (if any), and the timestamp of the lock through the retrieve option, `DKRetrieveOptions::basePropertyCheckedOutDetails(Boolean)`. See the detailed information for this method in the *Application Programming Reference* for information about usage, behavior, and in which DDO properties the information is stored. You can also call `DKDatastoreExtICM` methods to check the current checked-out status and find the current lock owner, but at the cost of a server call.

To check individual items in and out, use the `checkIn` and `checkOut` methods of the `DKDatastoreICM` class, as shown in the following steps. You can combine checkout with retrieve or combine checkin with update.

1. Connect to a `DKDatastoreICM` object named `dsICM` and use a DDO named `myDDO` to check out the item and check it back in. Normally, you perform additional operations between checkout and checkin.

   Java

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

   C++

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

2. If you want to combine checkout with retrieve and combine checkin with update, see the following example. Normally you perform operations between retrieve and update. If required, you can obtain the checked-out status to 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 options, `DKRetrieveOptionsICM::functionCheckout(true)`. The following example reuses the same options from 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, see the `DKRetrieveOptionsICM::functionCheckOut(Boolean)` method in the *Application Programming Reference*.

For a demonstration on checking items in and out, see the `SItemUpdateICM` C++ code 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 IBM 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.

The following 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. The version control policy can have one of the following values:

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);
```

Change SMS method

During the re-indexing of an item, you cannot change the resource manager or the collection. This change can only be made when you add document parts or resource items. Use the ChangeSMS method to change the collection for resource items and document model parts.

After an item is re-indexed, you can change the collection only, not the resource manager. To change the collection, there are two different procedures you can use to make changes for resource items and document model parts.

For resource items

Use any of the following code options:

Option 1

```
DKStorageManageInfoICM storageInfo =
(DKStorageManageInfoICM)lob.
   getExtension("DKStorageManageInfoICM");
storageInfo.setCollectionName(targetCollectionName);
lob.changeStorage();
```

Option 2

```
lob.changeSMSInfo(targetCollectionName);
```

For document model parts

```
public DKDDO updateDoc(DKDDO ddoDocument)
  throws DKException,Exception {
    String itemtype = ((DKPidICM)ddoDocument.getPidObject()).
          getObjectType();
    DKItemTypeDefICM ITDef =
   (DKItemTypeDefICM)dsDefICM.retrieveEntity(itemtype);
    updateValuesToAttrsInDDO(ddoDocument, ITDef);  //(code not shown)
    changeSMSForParts(ddoDocument);
    ddoDocument.update();
    return ddoDocument;
   }
 public void changeSMSForParts(DKDDO ddo)
 throws DKUsageError,Exception{
   short dataId = ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
                             DKConstant.DK_CM_DKPARTS);
   DKParts parts = (DKParts)ddo.getData(dataIbd);
   dkIterator iter;
   DKLobICM part;

   int partsCardinality = parts.cardinality();
   if (parts != null)
      iter = (DKSequentialIterator)parts.createIterator();
   else
      return;
   while (iter.more()) {
      part = (DKLobICM) iter.next();
      DKStorageManageInfoICM storageInfo =
     (DKStorageManageInfoICM) part.getExtension("DKStorageManageInfoICM");
      String rmName = part.getRMName();

      if (rmName != null) {
         DKRMConfigurationMgmtICM rmCfg = getRMConfigurationMgmt();
         DKResourceMgrDefICM rmDef = rmCfg.retrieveResourceMgr(rmName);
         if (rmDef != null) {
             storageInfo.setCollectionName(targetCollectionName);
         }
      }
      part.setChangeSMSRequested(true);
   }
 }
```

Setting table spaces for item or component types

As an alternative to setting table spaces using the system administration client, you can program your application to set table spaces using the Java or C++ APIs.

DB2 only: To put index and LOB attributes into table spaces that differ from the table's table space, the table's table space must be database managed rather than system managed. Because IBM Content Manager uses a system managed table space (ICMLFQ32) by default, you cannot set an index or LOB attribute's table space when the table's table space is not set. LOB attributes must be put into a large DB2 table space.

Oracle only: Oracle supports multiple index and LOB table spaces.

To retrieve and set table spaces, complete the following steps:

1. Connect to the content server using the `DKDatastoreICM.connect()` method. For dynamic table spaces, use a connection string of `DYNAMICTABLESPACEINFO=TRUE`. The connection string allows the getitemtype or listindexoncomp stored procedure calls to return dynamic table space information. Otherwise, the `DKDatastoreICM.connect()` method assumes static table spaces and defaults to `DYNAMICTABLESPACEINFO=FALSE`. For static table spaces, the table space values for the component type are stored in the following ICMSTCOMPDEFS columns: TABLETABLESPACE, DFTINDEXTABLESPACE, and DFTLOBTABLESPACE.

   Note that the library server configuration setting can override this option. Also, the option only affects display and XML export (not item type creation or the XML import).

   Alternatively, you can enable dynamic table spaces by calling `DKLSCfgDefICM.setRetrieveDynamicTablespaceInfoDisabled(boolean bTurnOn)`. To verify whether DYNAMICTABLESPACEINFO has been disabled or not, call `DKLSCfgDefICM.isRetrieveDynamicTablespaceInfoDisabled()`.
2. Whenever you create or update an item type to add component type, you can call one of the following APIs to set table spaces at the component type level (item type is considered the root component type):

Option	Description
API	Purpose
DKComponentTypeDefICM.setTable space(String)	Sets the table table space, which is the table space for creating the table of the component type.
DKComponentTypeDefICM.setIndexTable space(String)	Sets the default index table space, which is the table space for the database indexes of the component type.
DKComponentTypeDefICM.setLOBTable space(String)	Sets the default large binary object (LOB) table space, which is the table space for the LOB attribute columns of the component type.

   DB2 always stores the indexes and LOB attributes according to the initial definitions of the table spaces, and makes all three table spaces read only after the item type or component type creation completes. Oracle allows the default index table space and the default LOB table space to be modified after the item type or component type creation completes (as the indexes and LOB attributes can be put to multiple table spaces).
3. To retrieve the table spaces, you can call one of the following APIs:

Option	Description
API	Purpose
DKComponentTypeDefICM.getTable space()	Retrieves the component type table space.
DKComponentTypeDefICM. getIndexTableTable space()	Retrieves the index table space.
DKComponentTypeDefICM.getLOBTable space()	Retrieves the large binary object (LOB) table space.
DKComponentTypeIndexDefICM.getTable space()	Retrieves the index table space.

For more information about tablespace APIs, see the *Application Programming Interface Reference*.

Working with hierarchical item types

Hierarchical item types provide a data modeling feature for storing items in a rooted hierarchical folder structure.

By using hierarchical item types, you can create a hierarchy of document and folder items that is similar to a conventional file system. A major benefit of using hierarchical item types is guaranteed unique file names within a given parent folder.

"Creating a hierarchical item type"
"Folders-only item types" on page 120
"Adding the ICM$NAME attribute to a hierarchical item type" on page 120
"Root hierarchical folder" on page 122
"System default hierarchical folder" on page 123
"Hierarchical item constraints" on page 125
"Constraints for creating hierarchical items" on page 128
"Constraints for reindexing hierarchical items" on page 130
"Constraints for linking hierarchical items" on page 132
"Constraints for deleting hierarchical items and documents" on page 135
"Constraints for updating hierarchical items and documents" on page 135
"Concurrency of hierarchical run time operations" on page 136
"Hierarchical folder model adoption strategy" on page 138

Related information

Hierarchical item operations using Web services

Hierarchical item operations using XML services

Sample to demonstrate using the Content Manager APIs to create hierarchical item types and items

Creating a hierarchical item type

You create a hierarchical item type by enabling the hierarchical option of an item type when it is created.

After it is enabled, the hierarchical option of an item type cannot be changed. The hierarchical option can be set only for a non-resource, resource, or document item type. When the hierarchical option of an item type is enabled, the library server enforces hierarchical constraints for its items and causes hierarchical feature-related metadata to be added. In addition, the ICM$NAME attribute is added to the root component type.

If an attempt is made to set the hierarchical option of a part item type, error ICM7370 is returned. If an attempt is made to change the hierarchical option of an item type, error ICM7371 is returned.

Items of a hierarchical item type (hierarchical items) have the following characteristics:

- A single root folder exists for all hierarchical items.
- A hierarchical item is required to have a name value.
- A hierarchical item, including folders, must have a single hierarchical parent folder (except for the hierarchical root folder, which does not have a parent folder).
- A hierarchical item (including folders) can always be reached from the root hierarchical folder.
- Within a given hierarchical parent folder, each hierarchical item name must be unique.
- A hierarchical parent folder can be deleted only if it is empty.
- Hierarchical item types with the folders-only option set to true can hold only hierarchical folder items (semantic type equals Folder).
- Hierarchical item types with the folders-only option set to false can hold only hierarchical non-folder items.

Specifying a parent folder

Because a parent folder is required it must be specified when you create a hierarchical item, unless the system is configured to use a default folder. The parent folder is specified during item creation using the `DKConstantICM.DK_ICM_PROPERTY_PARENT_FOLDER` property. For example:

```
DKDDO parent = dsICM.getRootFolder();
propId = ddo.addProperty
   (DKConstantICM.DK_ICM_PROPERTY_PARENT_FOLDER,parent);
```

Hierarchical and non-hierarchical items

Hierarchical and non-hierarchical items cannot coexist in the same hierarchical item type. Hierarchical items are items, documents, and folders that created in a hierarchical item type, which is an item type that has the hierarchical option enabled. Non-hierarchical items refer to regular items, documents, and folders created in an item type that has the hierarchical option disabled. See the following table for the behavior of hierarchical and non-hierarchical item types in the same hierarchical item type.

Table 13. Hierarchical and non-hierarchical items

Runtime operation	Behavior of hierarchical and non-hierarchical items together
Manual linking	A hierarchical item and a non-hierarchical one can link together with any link type. For example, a hierarchical parent folder can contain both hierarchical and non-hierarchical items. However, they are not considered to be a hierarchical link. A link is considered as a hierarchical link only if both parent and child are hierarchical items, the semantic type of parent is Folder, and the link type is DKFolder. Hierarchical constraints are enforced when operating a hierarchical link.

Table 13. Hierarchical and non-hierarchical items (continued)

Runtime operation	Behavior of hierarchical and non-hierarchical items together
Auto linking	A hierarchical item type can be the target item type only of an auto-linking rule. In addition, the source item types of a hierarchical item type must be non-hierarchical item types. When you create a hierarchical item in the target item type, non-hierarchical folders are propagated and linked as normal.
Reference attributes	A hierarchical item can be referenced by a non-hierarchical item as the reference attributes, and vice versa.
Foreign keys	You can define foreign keys on one or more user attributes (including ICM$NAME) of a hierarchical item type referencing one or more user attributes of a non-hierarchical item type, and vice versa.
Move (reindex) item	A hierarchical item can be moved to a non-hierarchical item type, and vice versa, as long it still satisfies the hierarchical constraints after the move operation.

Auto linking considerations

When a hierarchical item type is used as the target item type of an auto-linking rule, its source item types must be non-hierarchical item types. When you create a hierarchical item in the target item type, non-hierarchical folders are propagated and linked as normal. To define auto-linking rules for an item type from the system administration client, open the item type properties and navigate to the Auto-Linking tab. This item type is the target item type of auto-linking rules. In the tab, expand the list of available source item types for defining auto-linking rules. To prevent usage errors, hierarchical item types are filtered from the list of available source item types for auto-linking rules in the system administration client. Only eligible source item types for auto-linking rules are listed. Ineligible item types that cannot be the source item type of an auto-linking rule are as follows:

- Document part item type
- Hierarchical item type

If you attempt to define auto-linking rules on a document part item type by calling the `addAutoLinkRule` method, error ICM7178 is returned. If you attempt to define auto-linking rules to use a hierarchical item type as the source item type by calling the `addAutoLinkRule` method, error ICM7372 is returned.

AutoLinkEnable option

The AutoLinkEnable option is used by the system administration client to filter the source item types of defining auto-linking rules. When it is enabled for an item type, the system administration client displays this item type as the available source item type for defining auto-linking rules. When a hierarchical item type is created, the auto-linking option is disabled by default. This option cannot be enabled. If you attempt to do so, error ICM7372 is returned. However, if you create a non-hierarchical item type that is not a part-item type from the system administration client, the auto-linking option is enabled by default.

Creating a hierarchical item type

The following code snippet demonstrates how to create a hierarchical item type:

```
DKItemTypeDefICM itemType = new DKItemTypeDefICM (dsICM);
...
itemType.setHierarchical(true);
...
itemType.add();
```

Related information

Sample to demonstrate using the Content Manager APIs to create hierarchical item types and items

Folders-only item types

A folders-only item type enforces the Folder semantic type for all its items.

The folders-only option is valid for both hierarchical and non-hierarchical item types, but it is not valid for part item types. If you attempt to enable the folders-only option for a part item type, error ICM7373 is returned. When you enable the folders-only option, the library server verifies that all existing items have the Folder semantic type. If any items without the Folder semantic type are found, the folders-only option of the item type cannot be enabled and the error ICM7375 is returned. If item versions are enabled for an item, the semantic type must be of type Folder for all the versions. The folders-only option of an item type can be disabled at any time, unless it has existing hierarchical folder items; if you attempt to disable the folders-only option in this case, error ICM7374 is returned.

Hierarchical folder item types

A hierarchical folder item type is a hierarchical item type with its folders-only option enabled. A hierarchical folder item type contains only folder items (items of the Folder semantic type). By contrast, a hierarchical non-folder item type contains non-folder items and documents only.

A hierarchical folder item type has all the characteristics of a hierarchical item type and follows all the hierarchical constraints. Hierarchical folders can be created only in hierarchical item types that are configured with the folders-only option. Conversely, all hierarchical non-folder items must be created in an item type that has the folders-only option disabled.

To enable the folders-only option of a hierarchical item type, call the `setFoldersOnly` method. To determine if a hierarchical item type has the folders-only option enabled, call the `foldersOnly` method.

Adding the ICM$NAME attribute to a hierarchical item type

The ICM$NAME attribute is the standard name attribute for an item type. It is a required attribute for hierarchical item types. If you do not add the ICM$NAME attribute to an item type when you enable its hierarchical option, it is automatically added when the hierarchical item type is created.

You can add the ICM$NAME attribute to the root component of any new or existing item type, except the part type class. The ICM$NAME attribute can be applied to documents, folders, or any data that is returned while performing a browse or a search. As the standard name attribute, the ICM$NAME attribute is useful for sorting and querying operations on item types, such as sorting results by name and resolving name-based paths.

The ICM$NAME attribute has the following characteristics:

- It is a predefined user attribute.
- It cannot be updated or deleted.

- Its value does not have trailing blanks
- Its data type is VARCHAR with length 256.
- It cannot be added to any user-defined attribute groups.
- When it is created, its attribute ID conforms to the user attribute ID generation rule (1 is added to the maximum). Therefore, its attribute ID is greater than or equal to 1000 and is different on different Content Manager EE systems.

When the ICM$NAME attribute is added to a hierarchical item type, it has the following default component attribute properties. Unless otherwise noted, these properties can be changed:
- Not nullable
- Not unique. This property cannot be changed.
- Not text-searchable
- Not represented
- No default value

When you provide a value for ICM$NAME, there is no restriction on the characters you can use. However, all the trailing blanks in the value are removed by the library server. Therefore, if you have provided an ICM$NAME value with trailing blanks, those trailing blanks are not visible in any search results. If you provide an ICM$NAME value with a string consisting entirely of blanks, its value is converted to an empty string and error ICM7016 is returned from the library server.

The following restrictions apply for an ICM$NAME attribute of a hierarchical item type. You cannot:
- Update the uniqueness property of the ICM$NAME attribute after it has been added to an item type. If you attempt to do so, the library server returns error ICM7123.
- Add the ICM$NAME attribute to a child component. If you attempt to do so, error DGL7351A (`DK_ICM_MSG_HIERARCHICAL_CHILD_COMPONENT_USE_ICMNAME_INVALID`) is returned.
- Add the ICM$NAME attribute to a user-defined attribute group. If you attempt to do so, error DGL7353A (`DK_ICM_MSG_HIERARCHICAL_ATTR_GROUP_USE_ICMNAME_INVALID`) is returned.
- Set the ICM$NAME attribute to be nullable while creating or updating a hierarchical item type. If you attempt to do so, error DGL7349A (`DK_ICM_MSG_HIERARCHICAL_INVALID_ICMNAME_NULLABLE`) is returned.
- Set the default value of the ICM$NAME attribute while creating or updating a hierarchical item type. If you attempt to do so, error DGL7350A (`DK_ICM_MSG_HIERARCHICAL_INVALID_ICMNAME_DEFAULT_VALUE`) is returned.

To get the ICM$NAME attribute value of an item type, call the `getName()` method of its `DKDatastoreICM` object. For more information, see the *Application Programming Reference*.

Related information

Adding the ICM$NAME attribute to a nonhierarchical item type

ICM$NAME attribute

Root hierarchical folder

The `ICMROOTFOLDER` folder is the root of all hierarchical folders and documents. It does not have a parent hierarchical folder,

ICMROOTFOLDER hierarchical folder

You cannot add the root hierarchical folder as a child folder of a hierarchical folder. If you attempt to do so, error ICM7379 is returned. However, the `ICMROOTFOLDER` folder can have a parent non-hierarchical folder. You cannot delete, modify, or reindex the `ICMROOTFOLDER` hierarchical folder. If you attempt to do so, error ICM7378 is returned. The only recommended run time operation is the link operation, which allows you to link any hierarchical folders and documents to an `ICMROOTFOLDER` hierarchical folder.

When an `ICMROOTFOLDER` hierarchical folder is created, it has the following property values:
- itemTypeID = 410
- componentTypeID = 410
- item ACL = RootFolderACL
- semanticType = 2 (Folder)
- ICM$NAME = ICMROOTFOLDER

ICM$FOLDER item type

`ICMROOTFOLDER` is a predefined system hierarchical folder of the `ICM$FOLDER` item type.

The `ICM$FOLDER` item type has the following properties:
- Non-resource item type with an ID of 410
- Hierarchical option is enabled
- Folders-only option is enabled
- Item type ACL is PublicReadACL
- Item-level ACL checking
- Versions are disabled
- Root component type ID = 410
- ICM$NAME attribute, with no default value

To get the root folder for all hierarchical items, call the `getRootFolder` method.

RootFolderACL access control

By default the access control list (ACL) of the hierarchical root folder, `ICMROOTFOLDER`, contains no rules, which means that only users with the ItemSuperAccess privilege can access it. You must add rules for other users to access it. RootFolderACL, which has an ACLCode value of 5, is a predefined ACL that manages the access to the hierarchical root folder. Administrator users can use the RootFolderACL to manage the authority of each user and group that access `ICMROOTFOLDER`. ACL scenarios are as follows:

- Adding an ACL rule to RootFolderACL for a user with a privilege set that includes link privileges allows that user to link hierarchical items to the `ICMROOTFOLDER` folder.
- Adding an ACL rule to RootFolderACL for a group with a privilege set that includes link privileges allows users in that group to link their hierarchical items to the `ICMROOTFOLDER` folder.
- If you enable public access from the system configuration, and there is an ACL rule to RootFolderACL for the ICMPUBLC group with a privilege set that includes link privileges, then users can link their hierarchical items to the `ICMROOTFOLDER` folder.
- An administrator user with the ItemSuperAccess privilege can access the `ICMROOTFOLDER` folder without adding an ACL rule to RootFolderACL.

Although an administrator user can assign RootFolderACL to any hierarchical or non-hierarchical item types or items, it is not recommended.

RootFolderACL is created with the following properties:
- The access control code is 5.
- For a new installation, `RootFolderACL` is added to three predefined administration domains: Super Domain, Default Domain, and Public Domain. For an upgraded installation, it is added to all existing domains.
- There is no predefined ACL rule.

Retrieving the root hierarchical folder

The following code snippet demonstrates how to retrieve the root hierarchical folder:

```
DKDatastoreICM dsICM = new DKDatastoreICM();
dsICM.connect(dbName, userid, pw, connectString);

DKDDO root = dsICM.getRootFolder();
```

System default hierarchical folder

`ICMSYSDEFAULTFOLDER` is a hierarchical folder that is used as the system default hierarchical folder when a hierarchical parent folder is unavailable. Enabling the system default hierarchical folder lets you avoid an error when a hierarchical parent folder is missing during a create or move operation.

The system default hierarchical folder (`ICMSYSDEFAULTFOLDER`) is a predefined hierarchical folder of the `ICM$FOLDER` item type. Its hierarchical parent folder is the root hierarchical folder (`ICMROOTFOLDER`). The system default hierarchical folder is system-defined and is created at installation time. Enabling the system hierarchical default folder affects both new and existing applications.

The `ICMSYSDEFAULTFOLDER` hierarchical folder has the following property values:
- itemTypeID = 410
- componentTypeID = 410
- item ACL = SysDefaultFolderACL
- semanticType = 2 (Folder)
- ICM$NAME = `ICMSYSDEFAULTFOLDER`.
- Hierarchical parent folder = `ICMROOTFOLDER`

To enable your application to use the system default hierarchical folder, call the `setHierarchicalDefaultFolderEnabled` method. To retrieve the system default hierarchical folder, call the `getDefaultFolder` method.

You cannot delete, modify, or reindex the system default hierarchical folder. If you attempt to do, error ICM7380 is returned. The only recommended run time operation is the link operation, which allows you to link any hierarchical folders and documents to an `ICMSYSDEFAULTFOLDER` hierarchical folder. You cannot move the `ICMSYSDEFAULTFOLDER` hierarchical folder from the `ICMROOTFOLDER` hierarchical folder to a child folder of another hierarchical folder. If you attempt to do so, error ICM7381 is returned.

Because the input hierarchical parent folder is required, if the hierarchical parent folder is missing during a create or move operation and the system default hierarchical folder has not been set, the library server returns the error ICM7382.

The following run time operations use the `ICMSYSDEFAULTFOLDER` folder:
- Creating an item in a hierarchical item type without providing the hierarchical parent folder.
- Moving an item from a non-hierarchical item type to a hierarchical item type without providing the hierarchical parent folder.

SysDefaultFolderACL access control

SysDefaultFolderACL is a predefined system access control list (ACL). SysDefaultFolderACL manages the access to the `ICMSYSDEFAULTFOLDER` folder. Administrator users can use this ACL to manage the authority of each user and group that accesses the `ICMSYSDEFAULTFOLDER` folder.

SysDefaultFolderACL has the following properties:
- The access control code is 6.
- For a new installation, RootFolderACL is added to three predefined administration domains: Super Domain, Default Domain, and Public Domain. For an upgraded installation, it is added to all existing domains.
- There is no predefined ACL rule.

You can use SysDefaultFolderACL as follows:
- Adding an ACL rule to SysDefaultFolderACL, for a user with a privilege set that includes link privileges, allows that user to link hierarchical items to the `ICMSYSDEFAULTFOLDER` folder.
- Adding an ACL rule to SysDefaultFolderACL, for a group with a privilege set that includes link privileges, allows users in that group to link their hierarchical items to the `ICMSYSDEFAULTFOLDER` folder.
- If public access is enabled from the system configuration, and there is an ACL rule to SysDefaultFolderACL for the ICMPUBLC group with a privilege set that includes link privileges, users are allowed to link their hierarchical items to the `ICMSYSDEFAULTFOLDER` folder.
- An administrator user with the ItemSuperAccess privilege can access the `ICMSYSDEFAULTFOLDER` folder without adding an ACL rule to SysDefaultFolderACL.
- An administrator user can assign SysDefaultFolderACL to any hierarchical or non-hierarchical item types or items, but it is not recommended.

Hierarchical item constraints

Constraints are enforced for hierarchical items during run time.

The following constraints are defined by the hierarchical model:

- Rooting constraint: A single root folder must exist for all hierarchical items.
- Name constraint: Each hierarchical folder or document requires a name.
- Parent constraint: Each hierarchical item, document, or folder (except the hierarchical root folder) requires a single hierarchical parent folder.
- Reachable constraint: Each hierarchical item, document, or folder must always be reachable from the root hierarchical folder.
- Same-name sibling constraint: The name of each hierarchical item or document must be unique within a given hierarchical parent folder
- Delete constraint: A parent folder cannot be deleted if it is not empty.
- Folders-only constraint: Each item in a folders-only item type must have a semantic type of Folder.

Rooting constraint

A single root folder (`ICMROOTFOLDER`) must exist for all hierarchical folders and documents. The root hierarchical folder cannot have a parent hierarchical folder and cannot be added as a child folder of a hierarchical folder. If you attempt to add the root hierarchical folder as a child folder, error ICM7379 is returned. However, the `ICMROOTFOLDER` folder can have a parent non-hierarchical folder. The `ICMROOTFOLDER` folder is protected; you cannot delete, modify, or reindex it. If you attempt to do so, error ICM7378 is returned. The only recommended run time operation is the link operation, which allows you to link any hierarchical folders and documents to an `ICMROOTFOLDER` hierarchical folder.

Name constraint

Each hierarchical folder and hierarchical document requires a name, which is stored in the ICM$NAME attribute. Because the ICM$NAME attribute does not have a default value and cannot be null, you must provide its value at run time. When you provide a value for the ICM$NAME attribute, all the trailing blanks in the value are removed. Therefore, if you have provided an ICM$NAME attribute value with trailing blanks, those trailing blanks are not visible in any search results. An ICM$NAME attribute value with a string consisting entirely of blanks is trimmed to an empty string, and error ICM7016 is returned from the library server. If a hierarchical folder or hierarchical document is created and the ICM$NAME attribute value is missing, the following error is returned: DGL5218A (`DK_ICM_MSG_ATTR_DATA_CANNOT_BE_NULL`).

Parent constraint

Each hierarchical item, document, or folder requires a single hierarchical parent folder (except the hierarchical root folder) with a semantic type of Folder.

Specifying a hierarchical parent folder

Because a parent folder is required, it must be specified when you create a hierarchical item, unless the system is configured to use a default folder. The parent folder is specified during item creation using the `DKConstantICM.DK_ICM_PROPERTY_PARENT_FOLDER` property. For example:

```
DKDDO parent = dsICM.getRootFolder();
propId = ddo.addProperty
   (DKConstantICM.DK_ICM_PROPERTY_PARENT_FOLDER,parent);
```

Hierarchical parent folder input
When you create or reindex an item, document or folder for a hierarchical item type, you can specify that a hierarchical parent folder is required. Doing so ensures that the hierarchical item satisfies the parent constraint as soon as it is created or moved. In such a case, if the hierarchical parent folder is missing after creating or reindexing a hierarchical item, error ICM7382 is returned.

System default folder option
For existing user applications, which might not have the interface to provide the hierarchical parent folder when creating or reindexing a hierarchical item, there is a system configuration System Hierarchical Default Folder option to allow the `ICMSYSDEFAULTFOLDER` folder to be the default hierarchical parent folder.

Hierarchical link
The hierarchical feature inherits the Content Manager EE linking model in which the hierarchical parent folder is the source and its child is the target. A link is considered to be a hierarchical link if both the parent and the child are hierarchical items, the semantic type of the parent is Folder, and the link type is DKFolder. When creating or reindexing an item, document, or folder for a hierarchical item type, if the provided parent folder is not a hierarchical folder, the error ICM7383 is returned. Although a hierarchical item can be linked with a link type other than DKFolder, if the link type is not DKFolder, the link is not considered to be a hierarchical link. If the link type is DKFolder, the parent must have a semantic type of Folder; otherwise, error DGL5145A (`DK_ICM_MSG_LINK_SOURCE_NOT_FOLDER_FOR_DKFOLDER`) is returned. For a hierarchical link, the library server does not need to check the semantic type of the parent hierarchical folder because it is always of type Folder.

Security checking
Linking privileges are checked for adding a link with link type DKFolder between a hierarchical parent folder and its hierarchical child item when a hierarchical item is created or reindexed. The move link action is equivalent to removing a hierarchical child item from its hierarchical parent folder and then adding the hierarchical child item to a new hierarchical parent folder. The following linking privileges are checked:

- ItemAddLink: The privilege checking is on the hierarchical parent folder to add a link.
- ItemLinkTo: The privilege checking is on the hierarchical parent folder.
- ItemLinked: The privilege checking is on the hierarchical child item.
- ItemRemoveLink: The privilege checking is on the original hierarchical parent folder to remove the link.

If you do not have the proper privileges, error ICM7215 is returned.

FileNet® Business Process Manager event logging
Because FileNet Business Process Manager events are logged for folder relationships with a link type of DKFolder, these event logs include hierarchical link events. If you subscribe to FileNet Business Process Manager "Add to Folder" event, this event occurs when adding a link with link type DKFolder between a hierarchical parent folder and a hierarchical child item when a hierarchical item is created or reindexed. In addition, a move link action generates two FileNet Business Process Manager events: “Remove from Folder” and “Add to Folder”.

Single hierarchical parent folder

Each hierarchical item, document, or folder can have only a single hierarchical parent folder. Cyclical links are not allowed. Each of the following run time operations attempts to create multiple hierarchical parent folders and therefore causes an error:

- Adding or moving a link with link type DKFolder to another hierarchical parent folder for a hierarchical item that already has a hierarchical parent folder: The library server returns error ICM7389.
- Reindexing an item to a hierarchical item type that has a hierarchical parent folder and another hierarchical parent folder is provided, or has two more hierarchical parent folders, or has hierarchical child items: The library server returns error ICM7391. If the input parent folder is the same as the existing hierarchical parent, no error is returned for the duplicate link.

Non-hierarchical parent for a hierarchical item

You can manually link a hierarchical item to a non-hierarchical parent folder by using the link operation. However, this operation can also be done by providing the input of a non-hierarchical parent folder for a hierarchical item or folder during a reindex operation. In such as case, the library server adds a regular link between the hierarchical item and non-hierarchical parent folder.

Parent for a non-hierarchical item

During the creation or reindexing of a hierarchical item, the input of the parent folder can also be a non-hierarchical item, document, or folder. If a non-hierarchical item, document or folder is provided, the semantic type of the parent must be Folder. Otherwise, error ICM7394 is returned. Linking privileges are checked and a link with link type DKFolder is added between a created or moved item, document, or folder and the parent folder. For reindexing to a non-hierarchical item, if the input parent folder is the same as the existing parent, no error is returned for a duplicate link error and there is no enforcement of hierarchical constraints. However, if you are adding another hierarchical parent folder and there are two or more hierarchical parent folders with link type DKFolder after this run time operation, this item cannot be moved to a hierarchical item type unless it is unlinked from these hierarchical parent folders. The system configuration option, System Hierarchical Default Folder, does not apply to a non-hierarchical item, document, or folder. The input of a parent folder is not allowed for a non-hierarchical and hierarchical item, document, or folder during the update. If it is provided, the error `DGL7355` is returned.

Reachable constraint

Each hierarchical item, document, or folder must always be reachable from the `ICMROOTFOLDER` hierarchical folder. The following run time operations cause hierarchical items to be unreachable from the `ICMROOTFOLDER` folder:

- Removing a hierarchical link between two hierarchical items with link type DKFolder. The library server returns error ICM7390.
- Reindexing a hierarchical folder in the middle of hierarchical chain. The library server returns error ICM7392.
- If moving a hierarchical item to new hierarchical parent folder forms a hierarchical link cycle, all hierarchical items in that cycle are not reachable from the root folder. The library server returns error ICM7390.

Same-name sibling constraint

Name values must be unique within the same hierarchical parent folder. If name values are duplicated during the following operations, the library server returns error ICM7388:

- Creating a hierarchical item, document, or folder and providing the ICM$NAME attribute value
- Updating the ICM$NAME attribute value. This value is enforced only on the latest version of a hierarchical item (when versioning is enabled).
- Reindexing an item to a hierarchical item type, regardless of whether you provide the hierarchical parent folder or ICM$NAME attribute value.
- Moving a hierarchical item to another hierarchical parent folder by moving a link operation.
- Deleting the latest version of a hierarchical item.

The same-name sibling constraint is enforced on the latest version of all hierarchical children within the same hierarchical parent folder regardless of whether a user has access. A user might have different ACL settings among different versions of a hierarchical item and might not have the access to view the latest version of a hierarchical item. Also, a user might have different ACL settings among different hierarchical children of a hierarchical parent folder, and might not have access to certain hierarchical children of a hierarchical parent folder. Therefore, it is possible for a user to receive error ICM7388 without being able to find any such a hierarchical document existing with that same name.

Delete constraint

A hierarchical parent folder cannot be deleted if it is not empty. Therefore, before a hierarchical parent folder can be deleted, all of its hierarchical child folders and documents must first be deleted. If you attempt to delete a hierarchical parent folder that is not empty, error ICM7393 is returned.

Folders-only constraint

Each item in a folders-only item type must have a semantic type of Folder. If you attempt to create or reindex an item that does not have a semantic type of folder, error ICM7386 is returned. When you update an item in a (hierarchical or non-hierarchical) folders-only item type, the semantic type Folder cannot be changed. If you attempt to do so, error ICM7387 is returned.

A hierarchical item type, which has the folders-only option disabled, can contain non-folder items and documents only. Error ICM7384 is returned for the following actions:

- Creating a hierarchical folder (semantic type Folder) in a hierarchical item type.
- Reindexing a folder with a semantic type of Folder to a hierarchical item type.

Constraints for creating hierarchical items

Constraints are enforced when you create hierarchical items and documents.

The following constraints are enforced at run time when you create a hierarchical item or document:

- Name constraint: Each hierarchical item or document requires a name.
- Parent constraint: Each hierarchical item or document requires a single hierarchical parent folder.

- Reachable constraint: The hierarchical item or document must be reached from the root hierarchical folder.
- Same-name sibling constraint: The name value of a hierarchical item or document must be unique within a given hierarchical parent folder.
- Folders-only constraint: Each item in a folders-only item type must have a semantic type of Folder.

Specifying a hierarchical parent folder

Because a parent folder is required, it must be specified when you create a hierarchical item, unless the system is configured to use a default folder. The parent folder is specified during item creation using the `DKConstantICM.DK_ICM_PROPERTY_PARENT_FOLDER` property. For example:

```
DKDDO parent = dsICM.getRootFolder();
propId = ddo.addProperty
   (DKConstantICM.DK_ICM_PROPERTY_PARENT_FOLDER,parent);
```

Security checking

To create a hierarchical item or document, you must have the ItemAdd privilege. In addition, you must have access with the linking privileges on both the input hierarchical parent folder and the created hierarchical item.

System-defined objects protection

You cannot create new items in the `ICM$FOLDER` item type.

Auto-folder considerations

According to auto-linking rules, a hierarchical item type can be a target item type and a non-hierarchical item type can be a source item type. Non-hierarchical folders are propagated and linked as normal when a hierarchical item is created in the target item type.

The following table compares constraints for creating hierarchical and non-hierarchical items

Table 14. Constraints for creating hierarchical and non-hierarchical items

Constraint	Hierarchical item	Non-hierarchical item
Parent folder input (System Hierarchical Default Folder option is disabled)	Required. No default hierarchical parent folder if you do not provide a hierarchical parent folder (default behavior).	Optional
Parent folder input (System Hierarchical Default Folder option is enabled)	Optional. Defaults to `ICMSYSDEFAULTFOLDER` folder when you do not provide the hierarchical parent folder.	Optional. The System Hierarchical Default Folder option does not apply.
Semantic type Folder is enforced for input parent folder	Yes	Yes
ICM$NAME input	Yes. There is no default value for ICM$NAME.	Not applicable
Hierarchical constraints enforcement	• Name constraint • Parent constraint • Reachable constraint • Same-name sibling constraint	No
Folders-only constraint enforcement	Yes	Yes

Table 14. Constraints for creating hierarchical and non-hierarchical items (continued)

Constraint	Hierarchical item	Non-hierarchical item
Auto-folder support	Yes, but a hierarchical item can be the target item.	Yes

Constraints for reindexing hierarchical items

When you reindex an item between two hierarchical item types or a hierarchical and nonhierarchical item type, constraints are enforced for the hierarchical parent and child items.

Reindexing an item from a nonhierarchical item type to a hierarchical item type

If an ICM$NAME attribute value does not already exist, you must provide it when you reindex an item from a nonhierarchical item type to a hierarchical item type. This value must satisfy the naming and same name-sibling constraints.

A nonhierarchical item type is not required to have a hierarchical parent folder, but a hierarchical item type is required to have a single parent folder. The single parent folder constraint is enforced when you reindex an item from a nonhierarchical item type to a hierarchical item type according to these four possible scenarios:

- The nonhierarchical item does not have a hierarchical parent folder. To satisfy the single parent folder constraint, you must provide a hierarchical parent folder when you reindex the item.
- The nonhierarchical item has a single hierarchical parent folder. If you do not provide a new hierarchical parent folder, the original parent folder is retained and the operation is successful. You cannot provide a hierarchical parent folder to a hierarchical item that is being reindexed between two hierarchical item types, unless the hierarchical parent folder is the same as the existing hierarchical parent folder. Otherwise, it violates the single parent folder constraint. A nonhierarchical parent folder is allowed.
- The nonhierarchical item has multiple hierarchical parent folders. This operation fails because the single parent folder constraint is violated. It makes no difference whether the hierarchical parent folder is provided or not.
- The nonhierarchical item has one or more hierarchical child items. This operation fails if its hierarchical children violate the single parent folder constraint.

Reindexing an item between two nonhierarchical item types

There is no enforcement of any hierarchical constraints. You can optionally provide a hierarchical parent folder.

Reindexing an item between two hierarchical item types

The item retains the same parents. A hierarchical parent folder cannot be provided to a hierarchical item that is being reindexed between two hierarchical item types, unless the hierarchical parent folder is the same as the existing hierarchical parent folder. Otherwise, it violates the single parent folder constraint. A nonhierarchical parent folder is allowed. If an ICM$NAME value is provided, its value must satisfy the same-name sibling constraint.

Reindexing an item from a hierarchical item type to a nonhierarchical item type
When reindexing a hierarchical item from a hierarchical item type to a nonhierarchical item type, there are two possible scenarios:

- The hierarchical item is at the end of a hierarchical chain. As a hierarchical document or an empty hierarchical folder that does not contain any folders or documents, it can be reindexed successfully to a nonhierarchical item type. After it is reindexed, it becomes a nonhierarchical item, and therefore, there is no enforcement of any hierarchical constraints after reindexing. You can optionally provide a parent folder.
- The nonhierarchical item has a single hierarchical parent folder. If the hierarchical parent folder is not provided, the original parent item is retained and the operation is successful. The hierarchical item is in the middle of a hierarchical chain, which is a hierarchical parent folder that contains one or more hierarchical items. If you attempt to reindex it to a nonhierarchical item type, the reachable constraint is violated. As a work-around, you can reindex its child items to another hierarchical parent folder before reindexing the item.

Folders-only constraint
When reindexing items to a hierarchical item type, you must have the same folders-only setting because the semantic type cannot be updated from Folder to non-folder, or vice versa. A hierarchical item type contains non-folder items and documents only while a hierarchical folder item type contains only folders.

Semantic type update
When reindexing a hierarchical item to a hierarchical (Folder or non-folder) item type, you cannot change the semantic type from Folder to non-folder, or vice versa. If you attempt to do so, error ICM7385 is returned. However, you are allowed to change the semantic type from non-folder to non-folder.

Security checking
To reindex an item to a hierarchical or nonhierarchical item, you must have the ItemMove privilege. In addition, if the parent folder is provided, you must have linking privileges on both the input hierarchical parent folder and the reindexed hierarchical item.

System-defined objects protection
You cannot reindex system-defined `ICMROOTFOLDER` or `ICMSYSDEFAULTFOLDER` hierarchical folders. You cannot reindex items to the `ICM$FOLDER` item type.

Auto-folder considerations
According to auto-linking rules, a hierarchical item type can be a target item type and a nonhierarchical item type can be a source item type. Nonhierarchical folders are propagated and linked as normal when a hierarchical item is created in the target item type.

The following table compares constraints for reindexing hierarchical and nonhierarchical items

Table 15. Constraints for reindexing hierarchical and nonhierarchical items

Source	Destination: hierarchical item	Destination: nonhierarchical item
Hierarchical item	When reindexing an item that has no hierarchical children, the following hierarchical constraints are enforced: • Name constraint • Parent constraint • Same-name sibling constraint • Folders-only constraint	There is no enforcement of any hierarchical constraints. You can optionally provide a parent folder input.
Nonhierarchical item	When reindexing an item, the following hierarchical constraints are enforced: • Name constraint • Parent constraint • Same-name sibling constraint • Folders-only constraint	• You can reindex the item if it is at the end of a hierarchical chain. You can optionally provide a parent folder. There is no enforcement of any hierarchical constraints. • Due to the reachable constraint, the operation is blocked if the item is in the middle of a hierarchical chain.

Constraints for linking hierarchical items

When you add, move, or remove a link between two hierarchical items, constraints are enforced.

Adding a link

When you add a link between two hierarchical items, the following rules must be followed:

- If you provide multiple links at the same time, the links cannot be duplicates unless you enable the Ignore Linking Error option. If there is link duplication and the Ignore Linking Error option is disabled, the library server returns the error ICM7015.
- You cannot manually link two hierarchical items together with a DKFolder link type, because this link type violates the single parenting constraint.
- You cannot manually add the `ICMROOTFOLDER` hierarchical folder as a child folder to a hierarchical folder.

Removing a link

Removing a hierarchical link violates the reachable constraint and it is not allowed. However, removing a non-hierarchical link is allowed. If you provide multiple links at the same time, the Ignore Linking Error option ignores deleted link not found errors.

Moving a link

The `moveToFolder` method moves a target item from the original source folder to a new source folder for the link type specified. Before calling this method, there must be an existing link between the original source folder and the target item with the specified link type. After the target item is moved to the new source folder, the link of the specified link type to the original source folder is deleted and a new link of the specified link type is created between the new source folder and the target item. If the Ignore Linking Error option is enabled, the removal of links that do not exist and the addition of duplicate links do not produce errors. If the original source folder, the new source folder, and the target item all belong to hierarchical item types and the link type is DKFolder, then the move must conform to hierarchical constraints. The Ignore Linking Error option does not ignore hierarchical constraint violations.

Security checking

The move link action is a shortcut for removing a target item from an original source folder and then adding the target item to a new source folder. Therefore, the linking privileges are checked for removing a link and adding a link separately as follows:

- ItemRemoveLink: The privilege checking is on the original source folder to remove the link.
- ItemAddLink: The privilege checking is on the new source folder to add a link.
- ItemLinkTo: The privilege checking is on the new source folder to verify that the user can link to it.
- ItemLinked: The privilege checking is on the target item.

FileNet Business Process Manager event logging

Because the move link action is considered to be a shortcut for removing a target item from an original source folder and then adding the target item to a new source folder, it generates two FileNet Business Process Manager events: “Remove from Folder” and “Add to Folder”. These events are based on the subscription of a user for moving the target item. The FileNet Business Process Manager events are logged only for folder relationships with the link type DKFolder and therefore also apply to hierarchical links.

Moving a link for a hierarchical item to switch the parent

For a hierarchical item, there are four scenarios for manually moving a link from one parent folder to another:

- Both the original and the new parent are hierarchical parents. If the link type is DKFolder, the move link request to switch the parent is successful. The same-name sibling constraint is enforced for the new hierarchical parent folder.
- The original parent is a non-hierarchical parent and the new parent is a hierarchical parent. If the link type is DKFolder, this move violates the single parenting constraint.
- The original parent is a hierarchical parent and the new parent is a non-hierarchical parent. If the link type is DKFolder, this move violates the reachable constraint.
- Both the original and the new parent are non-hierarchical parents. Because this link is a non-hierarchical link, no constraints are enforced and the move link request is successful.

Hierarchical link cycle prevention

If you switch from the original hierarchical parent folder to a new hierarchical parent that is one of the hierarchical children or descendants of this hierarchical item, it forms a hierarchical link cycle and disconnects from the hierarchical root folder, which violates the reachable constraint. Because the single parent constraint is supported for each hierarchical item and folder, it is not possible to form a hierarchical link cycle and still connect to the hierarchical root folder. A hierarchical item cannot point to a hierarchical parent folder that eventually reaches the hierarchical root folder and one of its hierarchical children or descendants.

System-defined objects protection

You cannot move a `ICMSYSDEFAULTFOLDER` hierarchical folder from a `ICMROOTFOLDER` hierarchical folder to another hierarchical folder

Summary table for comparing add and remove link operations

The following table compares add link and remove link operations for different kinds of link types.

Table 16. Add link and remove link operations for different link types

Link type	Add link	Remove link
Hierarchical parent to hierarchical child	• The operation fails if the link type is DKFolder, due to the single parenting constraint. • The operation succeeds if the link type is not DKFolder, because the link is not considered to be a hierarchical link.	• The operation fails if the link type is DKFolder, due to the reachable constraint. • The operation succeeds if the link type is not DKFolder, because the link is not considered to be a hierarchical link.
Non-hierarchical parent to hierarchical child	The operation succeeds, regardless of link type, because the link is not considered to be a hierarchical link.	The operation succeeds, regardless of link type, because the link is not considered to be a hierarchical link.
Hierarchical parent to non-hierarchical child	The operation succeeds, regardless of link type, because the link is not considered to be a hierarchical link.	The operation succeeds, regardless of link type, because the link is not considered to be a hierarchical link.
Non-hierarchical parent to non-hierarchical child	The operation succeeds, regardless of link type, because the link is not considered to be a hierarchical link.	The operation succeeds, regardless of link type, because the link is not considered to be a hierarchical link.

Summary table to compare moving a link to switch the parent

The following table compares moving a link to switch the parent for different link types.

Table 17. Moving a link to switch the parent

Move link	Link type is DKFolder	Link type is not DKFolder
• Child: hierarchical • Original parent: hierarchical • New parent: hierarchical	The operation succeeds and the same-name sibling constraint is enforced.	The operation succeeds because the link is not considered to be a hierarchical link.
• Child: hierarchical • Original parent: hierarchical • New parent: non-hierarchical	The operation fails, due to the reachable constraint.	The operation succeeds because the link is not considered to be a hierarchical link.

Table 17. Moving a link to switch the parent (continued)

Move link	Link type is DKFolder	Link type is not DKFolder
• Child: hierarchical • Original parent: non-hierarchical • New parent: hierarchical	The operation fails, due to the single parenting constraint.	The operation succeeds because the link is not considered to be a hierarchical link.
• Child: hierarchical • Original parent: non-hierarchical • New parent: non-hierarchical	The operation succeeds because the link is not considered to be a hierarchical link.	The operation succeeds because the link is not considered to be a hierarchical link.
• Child: non-hierarchical • Original parent: hierarchical • New parent: hierarchical	The operation succeeds because the link is not considered to be a hierarchical link.	The operation succeeds because the link is not considered to be a hierarchical link.
• Child: non-hierarchical • Original parent: hierarchical • New parent: non-hierarchical	The operation succeeds because the link is not considered to be a hierarchical link.	The operation succeeds because the link is not considered to be a hierarchical link.
• Child: non-hierarchical • Original parent: non-hierarchical • New parent: hierarchical	The operation succeeds because the link is not considered to be a hierarchical link.	The operation succeeds because the link is not considered to be a hierarchical link.
• Child: non-hierarchical • Original parent: non-hierarchical • New parent: non-hierarchical	The operation succeeds because the link is not considered to be a hierarchical link.	The operation succeeds because the link is not considered to be a hierarchical link.

Constraints for deleting hierarchical items and documents

When you delete a hierarchical item or document, hierarchical constraints are enforced.

Before you can delete a hierarchical parent folder, you must first delete all of its documents and hierarchical child folders. Otherwise, error ICM7393 is returned. If the hierarchical folder contains only non-hierarchical entities, or the link type is not DKFolder, the folder can be deleted because the link is not considered to be a hierarchical link.

Security checking
You must have the ItemDelete privilege to delete a hierarchical item, document, or folder. Although the deleted hierarchical item is removed from hierarchical and non-hierarchical parent folders, linking privileges are not checked for the delete action.

System-defined objects protection
You cannot delete the `ICMROOTFOLDER` or `ICMSYSDEFAULTFOLDER` hierarchical folder.

Constraints for updating hierarchical items and documents

When you update a hierarchical item or document, hierarchical constraints are enforced.

The following constraints are enforced when you update a hierarchical item or document:

- Name constraint: Each hierarchical folder or document requires a name.
- Parent constraint: Each hierarchical item, document, or folder (except the hierarchical root folder) requires a single hierarchical parent folder.
- Same-name sibling constraint: The name value of a hierarchical item or document must be unique within a given hierarchical parent folder
- Folders-only constraint: Each item in a folders-only item type must have a semantic type of Folder.

Version considerations
When there are multiple versions of a hierarchical folder or hierarchical document, the latest version is used for same-name sibling checking. If the current version is deleted, the next latest version is checked, and so on.

Semantic type update
For a hierarchical item in a hierarchical (Folder or non-Folder) item type, you cannot update the semantic type from Folder to non-Folder, or vice versa. If you attempt to change the semantic type from Folder to non-Folder, error ICM7387 is returned. If you attempt to change the semantic type from non-Folder to Folder, error ICM7385 is returned.

Security checking
You must have the ItemSetUserAttr privilege to update the ICM$NAME attribute or semantic type (if allowed) for a hierarchical item, document, or folder. Because the hierarchical parent folder is not changed during the update action, linking privileges are not checked.

System-defined objects protection
You cannot modify the `ICMROOTFOLDER` or `ICMSYSDEFAULTFOLDER` hierarchical folder.

Auto-folder considerations
According to auto-linking rules, a hierarchical item type can be a target item type and a non-hierarchical item type can be a source item type. Non-hierarchical folders are propagated and linked as usual when a hierarchical item is updated in the target item type.

Concurrency of hierarchical run time operations

When there are two or more run time operations involving hierarchical items operating at the same time, concurrency problems can occur. Concurrency problems can result in inconsistent hierarchical data.

Concurrency problems occur because there are items involved in the linking process that are not required to be verified because they do not change user data. To achieve high performance on run time operations, these concurrency problems (and the resulting inconsistent hierarchical data) are tolerated because they occur infrequently.

The following hierarchical operations are especially prone to concurrency problems:

- Changing the hierarchical state from hierarchical to non-hierarchical (or vice versa).

  For example, you are reindexing a non-hierarchical item to a hierarchical item type. Because this item does not have a hierarchical parent folder, you provide the hierarchical parent folder as part of the reindex input. At the same time, another user is adding the same item into a hierarchical parent folder.

- Adding or removing the folder link (link type is DKFolder) to an item.
- Deleting an item.

  For example, a user is deleting an empty hierarchical folder at the same time as another user is moving a hierarchical child item from a hierarchical folder to the folder being deleted. The result is a hierarchical link with a deleted hierarchical parent folder.

Avoiding concurrency problems

You can avoid concurrency problems and inconsistent hierarchical linking data by following these rules:

- Use reindex operations cautiously to change hierarchical state of an item or folder (from non-hierarchical to hierarchical or vice versa). Do not combine this type of operation with other more common operations such as create, delete, or link. Batch all moved items for changing hierarchical state and run reindex operation after hours.
- If you want to ensure that the items involved with a linking process are not deleted at the same time, manually check out these items, even though their data is not being changed. Items affected by this scenario include the parent folder of a created or reindexed item and both ends of a manually added link. Checking out these items serializes the items with other run time operations.

Symptoms and restoration of inconsistent hierarchical linking data

In general, you can search for inconsistent hierarchical linking data by using a query. This data can violate hierarchical single parent constraints, reachable constraints, or same name sibling constraints. The most common symptoms of inconsistent hierarchical linking data caused by concurrency problems are:

- The hierarchical link exists but one end of the link is no longer a hierarchical item or has been deleted. This situation can be detected by querying a hierarchical item or folder to find its hierarchical parent folder and being unable to find the item by searching for its ICM$NAME attribute. This situation violates the reachable constraint.
- A manually added or removed link is initially thought to be a non-hierarchical link, but is actually a hierarchical link (because the hierarchical state of one end of the link has been changed). This situation can be detected by querying a hierarchical item or folder to find its hierarchical parent folder, filtering by the ICM$NAME attribute. The query returns either no hierarchical parent folders or two or more hierarchical parent folders. When the hierarchical parent folder cannot be found, the reachable constraint is violated. When there are two or more hierarchical parent folders, the single parent constraint is violated. If two or more hierarchical children are found that have the same ICM$NAME attribute value, the same-name sibling constraint is violated.

After you find the inconsistent hierarchical linking data, you can restore the hierarchical link by performing the following recovery operations:

- If you cannot find the hierarchical parent folder of a hierarchical item, you can call the move link function by providing the hierarchical item with no hierarchical parent folder to be recovered and a valid hierarchical parent folder. If there is an existing valid hierarchical parent folder, the library server returns the error ICM7389. Restoring a non-hierarchical link for a hierarchical or non-hierarchical item by using move link function is allowed. However, if a hierarchical item, after restoring the link, still does not have a valid hierarchical parent folder, the library server returns the error ICM7390. You must first restore

the hierarchical link by providing a valid hierarchical parent folder before you can perform a move link action with a non-hierarchical parent folder as the new parent input for a hierarchical item.
- If you have two or more hierarchical parent folders or two or more hierarchical children that have the same ICM$NAME attribute value, you can call the remove link function by providing the suspected bad hierarchical link. If the link is identified as a bad hierarchical link (for example, the linking data is missing or the parent folder does not exist, or is no longer a hierarchical folder), it is removed. If the link is identified as a valid hierarchical link, the library server returns the error ICM7390.
- The most serious case is if you discover, after removing all the bad hierarchical links, that there are no valid hierarchical links. If this case occurs, you must verify that all the bad hierarchical links have been removed and then call the move link function with a valid hierarchical parent folder.

If you encounter inconsistent hierarchical linking data caused by concurrency issues, you can recover hierarchical links by passing an empty string or a nonexistent item to the remove link and move link function. For the remove link function, the parent, the child, or both can have an invalid itemID. For the move link function, only the original existing parent folder can have an invalid itemID. During the move link operation, if an existing valid hierarchical link is found and you provided a valid parent folder input that did not match the existing parent folder for this hierarchical link or you did not provide a valid parent folder, the library server returns the error ICM7396.

Related information

Explicit transaction behavior

Hierarchical folder model adoption strategy

The hierarchical folder model is flexible and provides advanced features and constraints. With the hierarchical folder model the overall performance for your applications is higher than if you managed your data by using the existing folder model and enforced the same hierarchical constraints.

The hierarchical folder model allows you to specify a name for each hierarchical folder and document and verifies that all folders and documents obey the hierarchical folder model rules. However, with the hierarchical folder model there is a performance penalty in the library server because of a table that it maintains. This table controls the same-name sibling constraint so that a name is unique within a given hierarchical parent folder. All run time operations on a hierarchical item impact this table by adding or removing data from it.

The hierarchical solution is provided to satisfy common and general folder requirements. You can continue to use the original folder model without sacrificing performance and introducing redundant data if you need only a simple link mechanism and do not require hierarchical features such as root folder, naming, and folder constraints. You can manually link any hierarchical or non-hierarchical item together.

When you use the hierarchical folder model in existing applications, there are two hierarchical inputs that your application must handle:
- Hierarchical parent folder. If an existing application does not have the interface to provide the hierarchical parent folder as an input, you can use the System Hierarchical Default Folder option to set the `ICMSYSDEFAULTFOLDER` folder as the default hierarchical parent folder.

- ICM$NAME attribute. This attribute requires a value and does not have a default. You must configure your application to allow the input of the ICM$NAME attribute value.

Auto-folder operations are restricted with the hierarchical folder model. A hierarchical item type cannot be the source item type of an auto-linking rule.

If you plan to move existing data to the hierarchical folder model, you can reindex non-hierarchical items to a hierarchical item type. Use caution when performing this operation. To avoid concurrency problems, do not use execute this procedure concurrently with other common operations such as create, delete, or link. It is recommended that you batch all moved items for changing the hierarchical state and reindex operation after hours.

The following table helps you decide whether to enable the hierarchical feature for an item type by reindexing.

Table 18. Hierarchical vs non-hierarchical item types

Requirement	Hierarchical item type	Non-hierarchical item type
Support basic folder and linking properties such as DKFolder link type and Folder semantic type.	Yes. The link type must be DKFolder and the semantic type must be Folder for hierarchical parent folders.	Yes. There is no restriction on link types and semantic types.
Support and enforce hierarchical constraints.	Yes. Hierarchical constrains are always automatically enforced.	No. You must develop and enforce constraints on demand.
Support and enforce folders-only constraint.	Yes	Yes
Provide new hierarchical inputs during run time operations.	Yes. Existing applications must be modified to provide new inputs. However, if an application cannot provide the hierarchical parent folder, there is a System Hierarchical Default Folder system configuration option to set the default parent folder to the `ICMSYSDEFAULTFOLDER` folder.	No. You do not need to change or configure your applications. However, you can use the interface of parent folder to create or move non-hierarchical items.
Run time performance impact	Yes. There is a performance penalty due to enforcement of hierarchical and folders-only constraints.	No. Your system performs the same as before. However, if the folders-only option is enabled, there is a performance penalty due to enforcement of folders only constraint.
Auto-linking and auto-folder feature support	Yes. Auto-folder feature is restricted.	Yes

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 mechanism 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 do not mean ownership. Therefore, the relationship of the folder to its contents is 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 alternative 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 alternative 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.

For IBM Content Manager Version 8.4.3, hierarchical item types are supported. A hierarchical folder item type is defined as a hierarchical item type with its folders-only option enabled.

The following procedures include code fragments for informational purposes only.

Important: Do not copy these code fragments and paste them directly into your application program because they do 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”
“Adding contents to a folder” on page 141
“Removing contents from a folder” on page 143
“Tips for adding and removing folder contents” on page 144
“Retrieving a folder's contents” on page 144
“Sorting the contents of a folder” on page 145
“Obtaining all folders containing a specific DDO” on page 146

Related information

Defining a folders-only item type

Creating a folder

The `DKDatastoreICM.createDDO()` method is used to create DDOs at run time.

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` `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.

The following steps show how to add contents to a folder. Afolder can contain another folder (subfolder) as one of its content items.

Important: You do not need to lock an item before updating it if you are only adding or removing links to the item without making any other changes in the DDO. However, if you need to change anything else in the DDO, such as modifying an attribute, you must check out the item before updating it.

"Adding contents to a folder example: Java"

"Adding contents to a folder example: C++" on page 142

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).

   Important: You do not need to lock an item before updating it if you are only adding or removing links to the item without making any other changes in the DDO. However, if you need to change anything else in the DDO, such as modifying an attribute, you must checkout the item before updating it.

   ```
   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);
   ```

   Important: Alternatively, you can combine the checkin request with your update call in Step 6 by using the `DKConstant.DK_CM_CHECKIN` update option.

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.

   Important: You do not need to lock an item before updating it if you are only adding or removing links to the item without making any other changes in the DDO. However, if you need to change anything else in the DDO, such as modifying an attribute, you must checkout the item before updating it.

   ```
   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);
   ```

Important: Alternatively, you can combine the checkin request with your update call in Step 6 by using the `DK_CM_CHECKIN(DKConstan.h)` update option.

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"

"Removing contents from a folder: C++"

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());
```

Sorting the contents of a folder

You can specify a sort function on the DKFolder collection to sort based on the created time stamp DDO property. Retrieve the attributes as optimally as possible by using multi-item retrieve. Use the DKRetrieveOptionsICM method and then use the DKFolder::sort(dkSort,boolean sortOrder) or the setSortFunction(dkSort) function. Pass an implementation object that extends the dkSort interface that determines how to sort.

To retrieve attributes for all folder contents, use multi-item retrieve on the DKFolder collection. For example:

```
// Suppose you have accessed the DKFolder collection "data item"
// value from your folder DDO
// given DKFolder in variable "dkfolder".
DKRetrieveOptionsICM dkRetrieveOptions =
 DKRetrieveOptionsICM.createInstance(dsICM);
dkRetrieveOptions.baseAttributes(true);
dsICM.retrieveObjects(dkfolder,dkRetrieveOptions.dkNVPair());

// Suppose you have a dkSort implementation in a class
//  named MySortSolution.
dkSort mySortSolution = new MySortSolution();
dkfolder.sort(mySortSolution,true);
```

Important: The choice of using the query or retrieve solution depends on the scaling factors. Initially, the query solution is a little faster than a very optimal use of retrieve and in-memory sorting. However, as the number of items and item types increase in your system, the retrieve solution becomes a better solution. Choose whichever is best for your current and future requirements.

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 the *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"
"Obtaining all folders containing a specific DDO: C++" on page 147

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 19. 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 the following table.

Table 20. 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"

"Link type names" on page 149

"Retrieving linked items" on page 149

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 20 on page 148, 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. Muti-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 IBM Content Manager system administration client or to the IBM Content Manager APIs for writing your own administration program, you can use the access control functions to control access to the information in your IBM 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" on page 151
"Creating a privilege set" on page 152
"Displaying privilege set properties" on page 153
"Defining an access control list (ACL)" on page 153
"Retrieving and displaying ACL information" on page 155
"Assigning an ACL to an item type" on page 156
"Assigning an ACL to an item" on page 157

"Using view ACLs on delete, add, or remove link operations" on page 157

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"

"Creating a privilege: C++"

Creating a privilege: Java

1. Connect to a datastore

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

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

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

3. Create a new privilege object.

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

4. Assign a name to the privilege object.

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

5. Assign a description to the privilege object.

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

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

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

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

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

8. Display information about the privilege object.

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

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("ICMNLSDB", "icmdmin", "password", "");
//Retrieve the datastore definition object (used to access
//and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore
//administration functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
            dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
                  dsAdmin->authorizationMgmt();
```

2. Create a new privilege object and set its properties.

   ```
   DKPrivilegeICM * priv = new DKPrivilegeICM(ds);
   //Set the name of the privilege object
   priv->setName("UserPriv");
   //Set the privilege description
   priv->setDescription("This is user-defined privilege");
   ```
3. Add the privilege to the datastore via the authorization management object.

   ```
   aclMgmt->add(priv);
   ```

Creating a privilege set

A privilege set is represented by the class `DKPrivilegeSetICM`.

Before you can begin creating privilege sets, you must be connected to a content server. To create a privilege set, complete the steps in the following topics:

“Creating a privilege set: Java”

“Creating a privilege set: C++”

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("ICMNLSDB", "icmdmin", "password", "");
//Retrieve the datastore definition object
//(used to access and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process
// datastore administration functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
             dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
             dsAdmin->authorizationMgmt();
```

2. Create three privileges and set their properties.

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

3. Create a new privilege set and set its properties.

```
DKPrivilegeSetICM * privSet1 = new DKPrivilegeSetICM(ds);
privSet1->setName("UserPrivSet");
privSet1->setDescription("This is a user-defined priv set");
```

4. Add the created privileges to the new privilege set.

```
privSet1->addPrivilege(priv_1);
privSet1->addPrivilege(priv_2);
privSet1->addPrivilege(priv_3);
```

5. Add the privilege set to the datastore using the authorization management object.

```
aclMgmt->add(privSet1);
```

Displaying privilege set properties

The example 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 IBM 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 an IBM Content Manager system must be bound to an ACL.

The following examples demonstrate how to define an ACL:

“Defining an access control list (ACL): Java”

“Defining an access control list (ACL): C++” on page 155

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("ICMPUBLIC");
   ```
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("ICMPUBLC");
   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);
   ```

   Important:

   If you update the ACL name in your language, the ACL name is updated for all the languages that are configured in IBM Content Manager instead of the default language only.
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("ICMPUBLIC");
   // 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");
   ```

   Important:

   If you update the ACL name in your language, the ACL name is updated for all the languages that are configured in IBM Content Manager instead of the default language only.
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"

"Assigning an ACL to an item: C++"

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

Each item type view is assigned its own access control list (ACL).

You can use a view ACL only when the ACL binding is either an item type or mixed with the item-level ACL flag set to false. Therefore, to use a view ACL, the ACL must be checked at the item-type view ACL instead of 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 access to the item to which you want to link.

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 also belongs to a second ACL (such as ACL 2) that gives the user read-only access to all items of an 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 IBM Content Manager.

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("ICMNLSDB","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;
DKLSConfigurationMgmtICM* lsCfg = NULL;
DKLSCfgDefICM* lsDef = NULL;
short systemFlag = 0;
long iSystemFlag = 0;
DKDatastoreICM* dsICM = new DKDatastoreICM();
dsICM->connect("ICMNLSDB","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 | 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.

The following sequence is the authentication flow path to implement single sign-on in your application:

1. A user enters their User ID and password through a browser application. 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. This cookie contains a Lightweight Third Party Authentication (LTPA) token.
4. The application logs on to IBM Information Integrator for Content by calling `connectWithCredential` in `DKDatastoreICM` and using the LTPA cookie. For 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("ICMNLSDB", credential, "");
   } else{
    dsICM.connect("ICMNLSDB", UserId, PWD, "") // normal connection with ID and PWD
   }
   ```
5. The IBM Information Integrator for Content API calls the 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 Content Manager EE.

The following figure illustrates the authentication flow path:

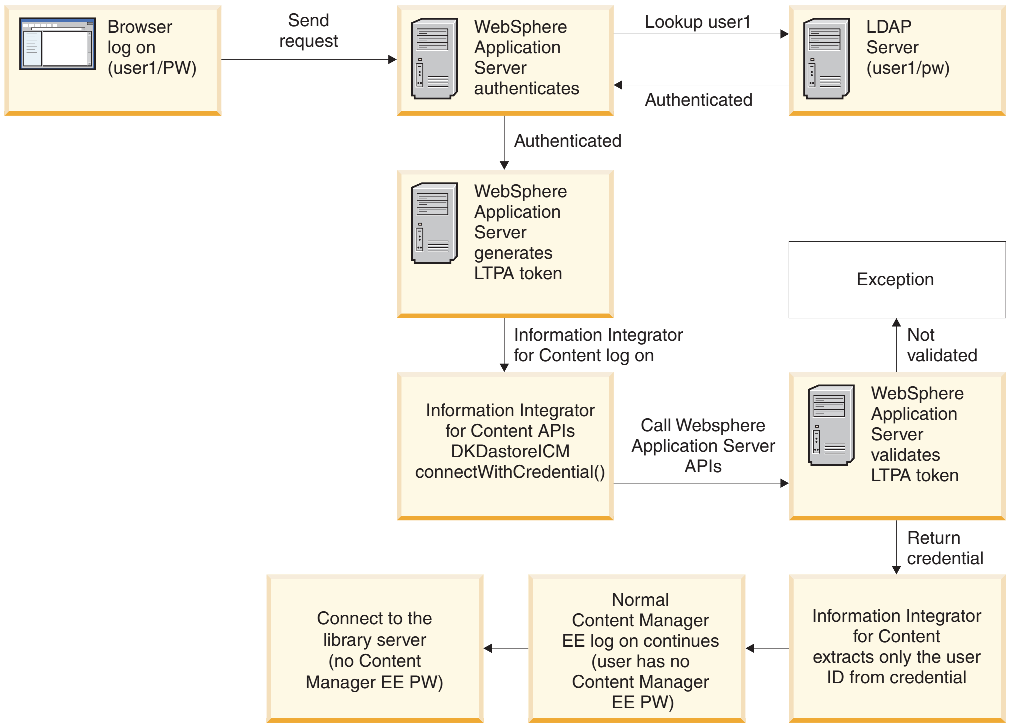

Browser log on (user1/PW)
Send request
WebSphere Application Server authenticates
Lookup user1
Authenticated
LDAP Server (user1/pw)
Authenticated
WebSphere Application Server generates LTPA token
Information Integrator for Content log on
Exception
Not validated
Information Integrator for Content APIs DKDastoreICM connectWithCredential()
Call Websphere Application Server APIs
WebSphere Application Server validates LTPA token
Return credential
Connect to the library server (no Content Manager EE PW)
Normal Content Manager EE log on continues (user has no Content Manager EE PW)
Information Integrator for Content extracts only the user ID from credential

Figure 8. Single sign-on authentication flow in a WebSphere Application Server environment

You can also use the `CMBConnection` bean to connect by using single-sign. The following example shows how to use 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 Content Manager (default is to use non-display names).
  connection.setDisplayNamesEnabled(true);

 }
```

To use single sign-on in the manner described in the previous section, configure your system to allow a trusted logon 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 **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 Content Manager EE from LDAP. This step is required because WebSphere Application Server authenticates the user with an LDAP server.
4. Ensure that the privilege set for all Content Manager EE users contains the AllowTrustedLogon privilege. To verify, open the properties panel of the user in the system administration client. Ensure that the privilege set for the user contains the AllowTrustedLogon privilege. For more information about privileges, see the *System Administration Guide*.

Library server user exits for IBM Content Manager

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

Important:
- For operating systems other than AIX, Solaris SPARC, Linux x86, Linux for System z®, all C user exits must be 32-bit library.
- For Linux for System z, all user exits must be 64-bit library, except the ICMFetchContentExit, which still must be 32-bit library.
- ICMACLPrivExit is a UDF and can be written as 32-bit or 64-bit library. The 64-bit UDF is supported when operating under DB2 64-bit instance.
- The library server user exits are supported both on DB2 and Oracle unless otherwise stated.

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 that can be used to build 32 or 64 bit user exits. Make sure to append the DB2 header file in the include path before you compile user exits such as icmaclxt.
- In the `bldUX.bat` script for Windows, append `/I%DB2HOME%\include` in the CL command, and replace %DB2HOME% with the DB2 instance home directory.
- In the `bldUX` script for UNIX, append `-I$INSTHOME/sqllib/include` in the MYSINC variable, and replace $INSTHOME with the DB2 instance home directory.

“Shared library names and location of the user exits”

“User exit ICMFetchContentExit for changing text” on page 163

“User exit ICMLogonExit for logon” on page 167

“User exit myExit for document routing” on page 169

“User exit ICMValidatePassword for validating passwords” on page 170

“ACL user exit routines” on page 173

Shared library names and location of the user exits

Each user exit must reside in a specific shared library, and the library must be copied to a specific directory.

The following table shows the shared library name and the directory where the user exits must be placed.

Table 21. User exit shared library and directory locations

User exit name	Shared library name	Directory path
ICMFetchContentExit (new)	**UNIX** `icmxlsfc0` **Windows** `icmxlsfc0.dll`	**UNIX** *ICMDLL*/*ls_dbname*/DLL **Windows** *IBMCMROOT*\cmgmt\ls\ls_dbname\DLL
ICMFetchContentExit (old)	**UNIX 32-bit** `icmdecxt` **UNIX 64-bit** `icmdecxt64 for ICMFetchContent, icmdecxt for ICMFetchFilter` **Windows** `icmdecxt.dll`	**DB2 on UNIX** *ICMDLL*/*ls_dbname*//DLL **Oracle on UNIX** *PATHICMDLL*/*ls_dbname* **Windows** *IBMCMROOT*\ls_dbname\DLL for ICMFetchContent, *IBMCMROOT*\cmgmt\ls\ls_dbname\DLL for ICMFetchFilter
ICMLogonExit	**UNIX** `ICMXLSLG.DLL` **Windows** `ICMXLSLG.DLL`	**UNIX** *PATHICMDLL*/*ls_dbname* **Windows** *PATHICMDLL*\ls_dbname
Document routing user exit	User-defined	The library path name and function are specified by the user when the node is defined.
ICMValidatePassword	**UNIX** `ICMPLSVP` **Windows** `ICMPLSVP.DLL`	**UNIX** *PATHICMDLL*/*ls_dbname*/DLL **Windows** *PATHICMDLL*\ls_dbname\DLL
ICMACLPrivExit	ICMACLXT (UDF) **UNIX** `ICMACLXT` **Windows** `ICMACLXT.dll (UDF)`	**DB2 on UNIX** *db2_installation_dir*/sqllib/function DB2 **Oracle PL/SQL** There is no shared library to copy. **Windows** *db2_installation_dir*\sqllib\function
ICMGenPrivExit	**UNIX** `ICMGENXT` **Windows** `ICMGENXT.dll`	**UNIX** *PATHICMDLL/ls_dbname/DLL* **Windows** *PATHICMDLL\ls_dbname\DLL*

The variables that are used in the previous table are:

ls_dbname
: The library server database name.

IBMCMROOT
: An environment variable that is set by the IBM Content Manager installation. It contains the path of installation directory, for example, `/usr/opt/db2cmv8` for AIX or `c:\Program Files\IBM\db2cmv8` for Windows.

PATHICMDLL
: The value of the column PATHICMDLL in the ICMSYSCONTROL library server table. The default value after installation is `IBMCMROOT\cmgmt\ls`. To

change value of the PATHICMDLL column, type the following SQL update command: `UPDATE ICMSTSYSCONTROL SET PATHICMDLL='newpath'`.

ICMDLL
A UNIX environment variable that users must set to `PATHICMDLL` by updating the `profile.env` file and the user profile in the DB2 instance.

Important: On UNIX, the permissions for the directory tree and the library itself must include the following user IDs:

DB2 The fenced ID of the DB2 instance.

Oracle The user ID of the operating system user who started the listener that services the library server instance.

Related information

Creating DB2 Universal database user IDs with user rights and privileges

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 new version of this user exit is supported on DB2 platform only. Since Fix pack 2, IBM 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.

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 IBM 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 be loaded only 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(int *psTIEFLAG, void *pvInputObject,
int  *plInputObjectLength,void *ppvOutputObject,
int *plOutputObjectLength, void  *malloc, int *plRC)
```

Input parameters

`int *psTIEFLAG`
Pointer to TIEFLAG value passed from Client

`char *pvInputObject`
Pointer to encrypted object sent to the exit

`int * plInputObjectLength`
Length of encrypted object

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

`int * plOutputObjectLength`
Length of decrypted object

`void * mallocf`
Memory allocation function passed to the user exit

Output parameters

`int *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 22. 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

Return codes

0 Object was converted in place 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 IBM Content Manager server log.

Related information

Recompiling your user exits from 32-bit to 64-bit

ICMFetchContentExit (new)

Since Version 8.3 Fix Pack 2, IBM Content Manager enhances the text search function by using the function ICMFetchContentExit.

IBM 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 as ICMFetchFilter.

The user exit function must reside in the `icmxlsfc.DLL` shared library for Windows, `icmxlsfc.a` library for AIX, and `icmxlsfc.so` library for UNIX. The shared library must be copied to the appropriate directory.

Example

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       */
                                   /* retrieve from the resource manager     */
   ICMRcTextReturned = 1,          /* Text returned - call INSO to convert   */
                                   /* Object will not be retrieved from RM   */
   ICMRcTextReturnedSkipInso = 4,  /* 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     */
                                   /* pfMalloc.  INSO conversion should not     */
                                   /* be called if MIMETYPE indicates text.   */

   ICMRcConversionError=2,         /* Error in conversion; No text returned   */
       /* to NSE; output object wriiten 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             */
                                        /* Used for consistency checking      */
   short  sExitVersion;              /* Version number of the exit.  The  */
                                        /* structure shown here is for V1.    */
   short  sExitType;                 /* 1: Pre-retrieve (called before    */
                                        /*    object is retrieved from RM)    */
                                        /* 2: Post-retrieve (called after     */
                                        /*    object was retrieved from RM)   */
   int    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

This user exit is supported on DB2 and z/OS. For Oracle, the user exit must be developed in the same way as in DB2 platform.

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

int **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:

int **plRC*
Return code

int **plReason*
reason code

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

int **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/icmxlslg.c`. For EIP, the code sample is in the EIP root directory in: `/samples/icmxlslg.c.`

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

Table 23. File locations

Sample name	Purpose
`icmxlslg.c`	C source code for user exit
`icmxlslg.def`	Standard `DEF` file for Windows platform
`icmnlslg.mak`	Make file for Windows(R) platform
`icmslslg.mak`	Make file for Sun platform
`icmxlslg.mak`	Make file for AIX platform

Return codes

The following list contains the return codes (*plRC) expected when logging on to IBM 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
 int   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 24. 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 IBM Content Manager library server user exit using Microsoft Visual Studio 2003

For document routine user exits, the path name of the shared library and the name of the function are specified by the users when creating the work nodes.

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. For Oracle, the user exit must be developed in the same way as in DB2 platform.

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 IBM 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. For Oracle, you must quiesce the library server and recycle `extproc`. For more information about how to quiesce the library server, see Preparing for fixpacks.

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/icmnlsdb/DLL`, the user exit is used by the library server to modify the password for IBM Content Manager users. However, if you are trying to modify password of IBM Content Manager administration user (icmadmin) from IBM 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

The 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
IBM 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 25. 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

Table 25. File locations (continued)

Sample name	Purpose
`bldUX.bat`	Compile IBM Content Manager library server user exit using Microsoft Visual Studio 2003
`bldUX`	UNIX script to compile IBM Content Manager library server user exit on AIX, Solaris, and Linux platforms

Return codes

The following list contains the library server return codes when you log in to IBM 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

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

“ICMACLPrivExit”

“ICMGenPrivExit” on page 176

ICMACLPrivExit

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 IBM Content Manager checks the privileges inside an SQL statement. This user exit is supported on DB2 and z/OS platforms. For Oracle, the implementation of this user exit is different.

Parameters

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

```
void SQL_API_FN ICMACLPrivExit(
      SQLUDF_CHAR     *pszUserID,        /* input */
      SQLUDF_CHAR     *pszApplicationID, /* input */
      SQLUDF_CHAR     *pszHostname, /* input */
      SQLUDF_INTEGER  * APIAction,       /* input */
      SQLUDF_CHAR     *pszProcedureName, /* input */
      SQLUDF_INTEGER  *InfoType,         /* input */
      SQLUDF_CHAR     *ItemID,           /* input */
      SQLUDF_INTEGER  *ViewID,           /* input */
      SQLUDF_INTEGER  *plGenPrivCode,    /* input */
      SQLUDF_INTEGER  *plRC,             /* output */
      SQLUDF_SMALLINT * pszUserID_ind,
      SQLUDF_SMALLINT * pszApplicationID_ind,
      SQLUDF_SMALLINT * pszHostname_ind,
      SQLUDF_SMALLINT * APIAction_ind,
      SQLUDF_SMALLINT * pszProcedureName_ind,
      SQLUDF_SMALLINT * InfoType_ind,
      SQLUDF_SMALLINT * ItemID_ind,
      SQLUDF_SMALLINT * ViewID_ind,
      SQLUDF_SMALLINT * plGenPrivCode_ind,
      SQLUDF_SMALLINT * plRC_ind,
      SQLUDF_TRAIL_ARGS_ALL
      )
```

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
Llibrary 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 26. 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

Note: DB2 on Windows will attempt to load UDFs using the PATH environment variable. Make sure that you do not have any other copy of the UDF in the directories listed in the PATH.

Return codes

0
RC_OK validation that the ICMACLPrivExit routine grants the ACL to query

Non-zero
Declines access to the ACL query

"Enabling the ACL user exit for IBM Content Manager on Oracle"

Enabling the ACL user exit for IBM Content Manager on Oracle:

To implement the user exit function in IBM Content Manager for Oracle, modify the sample user exit program and enable the access control list exit routines by using the system administration client.

To enable the sample user exit for Oracle:

1. Run the `icmaclxt_plsql.sql` file as the IBM Content Manager library server administrator user to create the PL/SQL user defined function. Example: `sqlplus <ls_admin>/<ls_password>@<ls_dbname> @icmaclxt_plsql.sql` where *ls_admin* is the actual library server database schema owner, *ls_password* is the password for *ls_admin*, and *ls_dbname* is the Oracle library server database name
2. Enable the access control list exit routines by using the system administration client:
   a. In the system administration client, select the appropriate library server.
   b. Open the Library Server Configuration window. Click the **Features** tab and select the **Enable ACL User Exit** check box.
3. Verify the sample user exit by performing the following test:
   a. Log in to SQL*Plus as the library server database administrator. Example: `sqlplus <ls_admin>/<ls_password>@<ls_dbname>`

b. Run the following SQL statement:

```
select ICMACLPRIVEXIT ('xyz','xyz','xyz',1,'xyz',1,'xyz',1,1)
from dual;
```

Expected Result: The SQL statement should return 0.

4. Modify the sample user exit program (`icmaclxt_plsql.sql`) to implement your own authentication logic.

Related information

Oracle error messages

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. For Oracle, the user exit must be developed in the same way as in DB2 platform.

Parameters

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

```
extern int ICMGenPrivExit(
  char *pszUserID,
  char *pszApplicationID,
  int *plGenPrivCode,
  char *pszProcedureName,
  int *plCMDecision,
  int *plCMDecReason,
  int *plRC
)
```

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
Llibrary server stored procedure name (20 characters)

pICMDecision
Library server decision

0 Grant access

Other codes
Look up library server error codes

pICMDecReason
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 27. 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

Return codes

0 RC_OK validation that the ICMGenPrivExit 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 an IBM Content Manager or federated database user ID that is not a database user ID, the connector tries and fails to connect to the database using that user ID. The connector then connects using the database connection user ID, and passes the IBM Content Manager or federated database user ID to the server.

To prevent the database connection from failing when using an IBM 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 so that it appears as follows: `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 an instance of the datastore has the behavior previously mentioned.

For an IBM Content Manager Version 8 database

In the `cmbicmsrvs.ini` file, update the ICMSERVERREPTYPE value to DB2CON instead of DB2 for a particular IBM Content Manager Version 8 database in the `cmbicmsrvs.ini` file, for example: `ICMSERVERREPTYPE=DB2CON`

After you change the ICMSERVERREPTYPE value, the IBM Content Manager connector logs on to the server using the database connection ID first, and then passes the IBM Content Manager Version 8 user ID to the server for that IBM Content Manager Version 8 database.

If the IBM 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 IBM Content Manager Version 8 user ID to log on to the database, and then passes the IBM 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 an instance of the datastore has the behavior mentioned previously.

Working with the resource manager

An IBM 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, extracting, encrypting, encoding, transcoding, searching, or text mining.

A single resource manager is used exclusively by one library server. Each resource manager delivered by the IBM Content Manager system provides a common subset of native data access APIs through which it is accessible by the controlling library server, by other IBM 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 IBM Content Manager component that must access content managed by IBM Content Manager 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 IBM Content Manager system as well as synchronous and asynchronous calls.

For information about how to configure a resource manager see the `SResourceMgrDefCreationICM` sample in the `samples` directory.

"Working with resource manager objects"

"Confidential retrieval of resource objects"

"Removing resource object contents" on page 180

"Asynchronous replication on a z/OS resource manager" on page 181

"Storing content in a user-specified collection by using Java or C++ API" on page 184

"API for resource manager" on page 186

Related information

The resource manager

Working with resource manager objects

Within IBM 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 IBM 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

IBM 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 `cmbcmenv.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 supported only for the IBM Content Manager Version 8 connector for objects stored on a fixed disk within a Windows, AIX, or Solaris resource manager.

Important: To use this function, the IBM Content Manager Version 8 connector, library server, and resource manager must be at the IBM Content Manager Version 8.2 fix pack level seven or later. If any of the IBM 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,DKRMSMSPairDefICM[] rmsmspairs)
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 IBM 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 IBM Content Manager. If objects were stored in IBM Content Manager before the creation and association of replication rules, IBM 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 `ICMMRMAP.JCL` to start the asynchronous replication process, ICMMOSAP. The input parameter list passed into ICMMOSAP from `ICMMRMAP.JCL` is then verified. The input parameter list is as follows:

   DB2SUBSYID (DB2C)
   : 4 bytes: The **`DB2SUBSYSID`** indicates the location where the z/OS resource manager database is located.

   IBM 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 IBM Content Manager user with system administrator authority.

   User Password (IFVTA1)
   : The IBM 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:

mapRMData()
: 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 it 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 definitions and column descriptions

Table 28. 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 29. ICMRMASYNCH

ICMRMASYNCH		
ITEMID	CHAR (26)	NOT NULL
VERSIONID	CHAR (5)	NOT NULL

Table 29. ICMRMASYNCH (continued)

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 IBM 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 30. 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 IBM 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, IBM 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

The IBM Content Manager Java API allows customers to remotely manage the resource manager.

IBM Content Manager provides an interface for applications to perform the following tasks:

- Configure IBM Tivoli® Storage Manager 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 resource manager API enhancements eliminate the need for system administrators to have physical access to the machine hosting the resource manager. In IBM Content Manager, the resource manager also supports WebSphere Application Server network clusters. The ability to manage the clusters in a network allows the resource manager to be managed from a remote location.

The IBM Content Manager resource manager allows you to configure the timeout length for service transactions. Configuring the timeout for service transactions improves the responsiveness of the resource manager services by allowing blocked items to be bypassed.

Important: The resource manager API enhancements are enabled by the ICM connector Java API. You must install IBM Content Manager Version 8.4 or later to use these API enhancements.

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

DKRMConstantSMS

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

Table 31. Constants in the DKRMConstantSMS class

Constants	Purpose	Value
`SMS_ALLOWED_MIN_LOW_THRESHOLD`	Indicates the minimum allowed value for a low threshold (resume threshold for enabling volume).	0
`SMS_ALLOWED_MAX_HIGH_THRESHOLD`	Indicates the maximum allowed value for a high threshold (threshold for disabling volume).	100
`SMS_ALLOWED_MIN_HIGH_THRESHOLD`	Indicates the minimum allowed value for a high threshold (threshold for disabling volume).	1

DKRMServerDefinitionDefSMS

The `DKRMServerDefinitionDefSMS` class has the following resource manager-related methods.

Table 32. Methods in the DKRMServerDefinitionDefSMS class

Methods	Purpose
`setDSMIConfig`	Sets the fully qualified path for the Tivoli Storage Manager client options file.

Table 32. Methods in the DKRMServerDefinitionDefSMS class (continued)

Methods	Purpose
`getDSMIConfig`	Returns the fully qualified path for the Tivoli Storage Manager client options file.
`setTSMBufferSize`	Sets the buffer size to use for data transfer between the resource manager and the Tivoli Storage Manager server.
`getTSMBufferSize`	Returns the buffer size to use for data transfer between the resource manager and the Tivoli Storage Manager server.
`setTSMFileSpaceName`	Sets the Tivoli Storage Manager file space name. Tivoli Storage Manager determines the buffer size of the content location that must be used for data transfer with the Tivoli Storage Manager server by using the `setTSMFileSpaceName` method.
`getTSMFileSpaceName`	Returns the Tivoli Storage Manager file space name. Tivoli Storage Manager determines the buffer size of the content location that must be used for data transfer with the Tivoli Storage Manager server by using the `getTSMFileSpaceName` method.
`setTSMFileSpaceInfo`	Sets an identifier of the resource manager objects within the Tivoli Storage Manager file space.
`getTSMFileSpaceInfo`	Returns an identifier of the resource manager objects within the Tivoli Storage Manager file space.

For more information, see the Javadoc reference documentation.

Example

The following example demonstrates the use of these `DKRMServerDefinitionDefSMS` class methods:

```
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.setDSMIConfig("/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 methods of the `DKRMVolumeDefSMS` class provide a flexible and granular way to control the migration of the objects.

Table 33. Methods in the DKRMVolumeDefSMS class

Methods	Purpose
`setHighThreshold`	Sets a high threshold (threshold for disabling volume) when 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) when the volume is considered full.
`setLowThreshold`	Sets a low threshold (resume threshold for enabling volume) when the 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 of the `DKRMVolumeDefSMS` class are deprecated:

setThreshold
Deprecated. Use the `setHighThreshold` and `setLowThreshold` methods instead.

getThreshold
Deprecated. Use the `getHighThreshold` and `getLowThreshold` methods instead.

For more information, see the Javadoc reference documentation.

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

The `DKRMControlServiceSMS` class has two `control` methods, which control resource manager services.

DKRMConstantSMS.CONTROL_SERVICE_RESPONSE control
This `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 returns a response that indicates the result or the status of the action performed.

```
DKRMConstantSMS.CONTROL_SERVICE_RESPONSE control
   ( DKRMConstantSMS.CONTROL_SERVICE_NAME service,
    DKRMConstantSMS.CONTROL_SERVICE_ACTION action)
```

DKControlServiceResponse control

This `control` method takes a service name, action, and timeout value as input parameters and sends an order to the resource manager to perform the action specified on the service indicated in the time specified. The resource manager returns a response class that indicates the result or the status of the action performed and the current timeout for the transaction.

```
DKControlServiceResponse control
(DKRMConstantSMS.CONTROL_SERVICE_NAME service,
 DKRMConstantSMS.CONTROL_SERVICE_ACTION action,
 int transactionTimeout)
```

For more information, see the Javadoc reference documentation.

Example

The following example demonstrates the use of the `DKRMControlServiceSMS` class `control` method that takes service name and action as input parameters:

```
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);
```

Example

The following example demonstrates the use of the `DKRMControlServiceSMS` class `control` method that takes service name, action, and timeout value as input parameters:

```
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
DKRMControlService response =
controlService.control( DKRMConstantSMS.CONTROL_SERVICE_NAME.migrator,
DKRMConstantSMS.CONTROL_SERVICE_ACTION.status, 0);
System.out.println("Response status = " + response.getStatus());
System.out.println("Response timeout = " + response.getTimeout());
```

DKRMConfigDefSMS

The methods of the `DKRMConfigDefSMS` class enable validation scheduling and allow validation-specific parameters to be set.

setValidatorMetadata

Sets the type of metadata to validate for the validation utility.

This method specifies one of the following `VALIDATION_METADATA` constants:

- `VAL_METADATA_ID_RMOBJECTS`: Validate metadata of resource manager objects.
- `VAL_METADATA_ID_RMMIGRATIONTASKS`: Validate metadata of resource manager migration tasks.
- `VAL_METADATA_ID_ALL`: Validate all metadata (resource manager objects and migrations tasks).
- `VAL_METADATA_ID_NONE`: Do not perform metadata validation.

```
public void setValidatorMetadata(DKRMConstantSMS.VALIDATION_METADATA)
```

getValidatorMetadata

Gets the type of metadata to validate for the validation utility.

This method returns one of the following `VALIDATION_METADATA` constants:

- `VAL_METADATA_ID_RMOBJECTS`: Validate metadata of resource manager objects.
- `VAL_METADATA_ID_RMMIGRATIONTASKS`: Validate metadata of resource manager migration. tasks
- `VAL_METADATA_ID_ALL`: Validate all metadata (resource manager objects and migrations tasks).
- `VAL_METADATA_ID_NONE`: Do not perform metadata validation.

```
public DKRMConstantSMS.VALIDATION_METADATA getValidatorMetadata()
```

For more information, see the Javadoc reference documentation.

DKControlServiceResponse

This class represents the reply information that is returned from a call to a `DKRMControlServiceSMS.control` method.

DKControlServiceResponse

A constructor for the response class returned from DKRMControlServiceSMS.control.

This constructor has the following parameters:

- *status* - The status of the control request that is processed by the resource manager service.
- *timeout* - The transaction timeout value that is set on the resource manager service.

```
public DKControlServiceResponse(
        DKRMConstantSMS.CONTROL_SERVICE_RESPONSE status,
        int timeout)
```

DKControlServiceResponse

A constructor for the response class returned from DKRMControlServiceSMS.control.

This constructor has the following parameters:

- *status* - The status of the control request that is processed by the resource manager service.
- *timeout* - The transaction timeout value that is set on the resource manager service.

- *extendedstatus* - The extended status information, if available. Otherwise, this value is `null`.

```
public DKControlServiceResponse(
         DKRMConstantSMS.CONTROL_SERVICE_RESPONSE status,
         int timeout,
         java.util.Map extendedstatus)
```

getExtendedStatus

Returns the extended status of the request as specified in the reply received from the resource manager service. Extended status contains name-value pairs containing additional information for the specific operation.

The validation service is the only service that returns extended status information. The following keys are included in the extended status:

- validator-current-operation: Reports the current step or operation that is active for the validation service.
- validator-estimated-count: Reports the estimated count of the items to be validated
- validator-retrieved-ls-items: Reports the number of validation items retrieved from the library server.
- validator-processed-items: Report the number of items that have completed metadata or storage validation operations.

Returns: The extended status, if available, for the requested action. Otherwise, returns `null`.

```
public Map getExtendedStatus()
```

getStatus

Returns the status of the request as specified in the reply received from the resource manager service.

Returns: The status for the requested action.

```
public DKRMConstantSMS.CONTROL_SERVICE_RESPONSE getStatus()
```

getTimeout

Returns the timeout value set on the resource manager service.

Returns: The timeout value in effect on the resource manager service. If the timeout is not used by specific service, -1 is returned. The timeout that is currently being used, if applicable, is always returned in the response.

```
public int getTimeout()
```

For more information, see the Javadoc reference documentation.

Managing documents in IBM Content Manager

IBM 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 IBM 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"

"Creating a document item type example: C++"

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 following steps take you through the process of creating an item (based on a predefined document item type) that contains one attribute and one part. 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”

“Creating a document example: C++” on page 196

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 following steps, a new part is added and the attribute value is updated.

"Updating a document example: Java"

"Updating a document example: C++"

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 item type of a document.

Document parts have the following versioning characteristics:

- 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 following examples show how to get the versioning rules for the base part of an item type.

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;
DKItemTypeRelationDefICM itemRelation =
(DKItemTypeRelationDefICM)item.retrieveItemTypeRelation(partId);
versionControlPolicy = itemTypeRelation.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
DKItemTypeRelationDefICM * itemTypeRelation =
  (DKItemTypeRelationDefICM*)itemType->retrieveItemTypeRelation(partId);
//Retrieve the version control policy for the specified part
int versionControlPolicy = itemTypeRelation->getVersionControl();
```

Working with transactions

Transactions allow IBM 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 IBM 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 IBM Content Manager system initiates a rollback when a severe error occurs or when it is necessary to resolve a deadlock between the library server and the database.

Within a transaction, uncommitted resource manager changes are not visible to the application that made the changes until the transaction is committed. For example, you make changes to a resource manager item and you store it. If you retrieve that item before the transaction is committed, the item will not reflect the changes that you just made. You will not see the updated item until the transaction is committed.

Concurrent or overlapping transactions through a single library server connection are not supported. To maintain concurrent transactions, you must make multiple connections between the library server and the database, or initiate multiple client processes or threads if you are working with a client application. Applications like IBM WebSphere ® Application Server handle processes, connections, and sessions.

The `execute()` and `executeWithCallback()` methods in `DKDatastoreICM` automatically create an additional connection to the database when invoked. The new database connection is then used to execute the query. Since queries use a separate database connection, they also have a separate transaction scope from other content server operations. The connection to the database is closed (or returned to the pool, if pooling is enabled) when the `DKResultSetCursor` is closed.

"Things to consider when designing transactions in your application"
"Caution when using transactions" on page 201
"Using checkin and checkout in transactions" on page 201
"Processing transactions" on page 202
"Explicit transaction behavior" on page 203
"Transaction behavior when deleting a user does not belong in a group" on page 203

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 if time prior to the transaction commit is the amount of time the user is locked out.

The best approach to avoiding database locking is to commit transactions often and to avoid long running transactions. If you must perform CRUD operations within a transaction, it is recommended that you perform these operations when it is understood that no one else will access the items being updated.

Using checkin and checkout in transactions

IBM Content Manager supports `check-out` and `check-in` operations on items.

The `checkout` 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 `checkout` 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 `checkin` 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 check out 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.

Important:

Calling the `checkout` method in an explicit transaction might cause contentions for other applications that are trying to check out the same items. As a result, the lock time out error (-911) might happen and the performance of the application might degrade. To avoid this contention, it is recommended that the applications perform the checkout action in an implicit transaction to alleviate system contentions for the resources. For applications that are not competing against the same item ID or for applications where the resource manager replicator is not running at the same time, calling the `checkout` method is not a problem.

If you check out or check in an item by 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. You must check the item back in by 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` class methods to complete all the transaction steps.
2. Call all of APIs that you want to include in the transaction in the order in which 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` method, and either the `commit` or `rollback` method, are treated as a single transaction.

All APIs can be included in transactions, unless specifically noted. See the *Application Programming Reference* for details. Some administrative APIs, such as the methods to define or update item types, cannot be included in transactions.

The following class methods are used in IBM Content Manager item creation and update transactions:

DKDatastoreICM.startTransaction()
Starts a 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 an 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()
: Reindexes an item. Moves an item from one item type to another item type. The item must be checked out prior to calling this method.

An additional source for information about transactions is the `SItemUpdateICM` sample.

Explicit transaction behavior

Beginning in IBM Content Manager Version 8.4, if the system detects an error during a persistent operation, the default behavior is to always automatically roll back an explicit transaction.

For example, XYZ Insurance has an online policy payment system, a user submits their credit card information for processing, but the power suddenly goes out. The credit card charge and the payment update to the policy are rolled back 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 stopped due to a power failure and you cannot undo the prior transactions. Your application might not remember what it was doing before the system stopped 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 are using explicit transactions for one or more run time operations within an IBM Content Manager transaction unit, it is recommended that you commit the transaction as soon as the document or folder is created. Doing so ensures the uniqueness of the item ID for your created document or folder and avoids possible transaction locking or timeout issues. In addition, because the IBM Content Manager checkout function cannot be part of an IBM Content Manager explicit transaction, you must commit the transaction as soon as you check out an item in order for other run time operations on the checked-out item to function correctly.

If you want to override the system 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 IBM 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 the IBM Content Manager data model, you compose queries by using the IBM 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, by using SORTBY.
- Enforces IBM 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 IBM 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 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). 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 on, your request is directed to the IBM 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 IBM Content Manager server" on page 206
"Applying the query language to the IBM Content Manager data model" on page 207
"Understanding parametric search" on page 209
"Writing queries by using predicates in path expressions" on page 210
"Understanding text search" on page 211
"Creating combined parametric and text search" on page 220
"Example searches by using the query language" on page 222
"Using escape sequences in your queries" on page 234

"Understanding row-based view filtering in query" on page 240

"The query language grammar" on page 247

Related information

Query performance guide

Querying the IBM 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 make the search more thorough. A combined query is composed of both text and parametric conditions.

When you query the IBM 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.

When you submit a query, you must submit a list of query options, each of which is documented in the *Application Programming Reference* (Javadoc) for the `DKDatastoreICM` class. One of the query options available includes an entry for retrieve options. Query calls multi-item retrieve (`DKDatastoreICM::retrieveObjects(dkCollection,DKNVPair[]`) method and passes the retrieve options you supplied (or the default options).

Important: It is recommended that you always specify the retrieve options instead of the default and use the `DKRetrieveOptionsICM` class. Submitting a `DKRetrieveOptionsICM` instance enables additional optimization and allows for better performance choices. For the C++ native API, it is especially important to always submit a `DKRetrieveOptionsICM` instance for better performance.

The `evaluate` method returns all results as a collection of DDOs. The default options work well for result sets that are relatively small, 200 items or less. When working with larger results sets, 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 because the data is needed by the application. You can limit the result set size by using the `max results` option.

The `execute` method returns a `dkResultSetCursor` object, which has the following characteristics:

- The `dkResultSetCursor` object works like a content server cursor.
- You can use the `dkResultSetCursor` object for large result sets because the DDOs it returns are retrieved in blocks as the user requests them.

- You can use the dkResultSetCursor object 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 object 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.

Important: The IBM Content Manager does not support scrollable cursors.

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 a description of the query options of the execute method.

Starting with IBM Content Manager Version 8 Release 3, there are two ways you can get the count of query results without retrieving the results themselves. One way is through the executeCount method in DKDatastoreICM, and the other way 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 IBM 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.

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 result occurs because the count is accurate only 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 is also inaccurate. To get an accurate count, a server call must be made to evaluate the query, which means that if you want the count and then the results, you must make two trips to the server.

For more information about 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 IBM 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 explaining the query language. Therefore, it is 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, IBM Content Manager data model elements are represented as follows:

Items In general, each IBM Content Manager 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 an IBM Content Manager component. The ITEMID and the VERSIONID are the same as the root component ITEMID and VERSIONID of the child.

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 IBM Content Manager data model, for the purpose of querying it is convenient to conceptually think of them as being a part of the XML element representing the item. This concept relieves applications from writing joins explicitly in the queries. You can use links to model a many-to-many relationship between items. 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 metadata 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 about 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 be a root component only, 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). References can be traversed in both directions.

Reverse traversal of references is performed in a way similar to reverse traversal of links, as described previously. 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 element 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 about reference attributes, see the `SReferenceAttrDefCreationICM` and `SSearchICM` samples.

Folders

Folder functionality provided in IBM Content Manager through the `DKFolder` is stored as a set of links of the `DKFolder(DK_ICM_LINKTYPENAME_DKFOLDER)` link type. Each folder-content relationship is modeled as an outbound link from the folder and an inbound link to the contents making the folder the source and the contents the targets of each link. Follow the semantics of searching and traversing links in order to search and traverse folders. For more information, see the `SFolderICM` and `SSearchICM` samples.

Understanding parametric search

Items are often retrieved by initiating a search on selected attributes. A single query can examine both system-defined and user-defined attributes of the items in the content server.

Simple search conditions consist of an attribute name, an operator, and a value that are combined into a clause. IBM 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 by using the Boolean operators `AND`, `OR`, and `NOT`. See the query examples for more details.

Writing queries by using predicates in path expressions

To improve the query run time and minimize the memory usage during run time, rewrite the XQPE query to an equivalent XQPE query by combining the predicates in the path expressions and reducing the SQL length.

To rewrite the XQPE query to an equivalent XQPE query, reduce the number of components in the XQPE query or path expression. Rewriting the XQPE query to an equivalent XQPE query is valid only when the predicates or the conditions that are being combined are connected with an OR operator, not with an AND operator.

Important: You must not use this method for queries that involve the AND operator. The rewrite has a different meaning and returns different results.

```
Q3: /A[B/@attr3 = "a" AND B/@attr4 = "b"]
Q4: /A[B[@attr3 = "a" AND @attr4 = "b"]]
```

Query `Q3` queries for item type `A` with a child component `B` in which attribute `attr3` is equal to "a" and a child component `B` with attribute `attr4` equal to "b". However, the query could be rewritten to query for different child components in which each child component satisfies one of the conditions. Query `Q4` queries for item type `A` with a single child component `B` that satisfies both conditions. Note that query `Q4` is not an equivalent rewrite; query `Q3` cannot be replaced with query `Q4` because the queries are different.

Example of rewriting a query

Item type `A` has attributes `attr1` and `attr2`. Item type `A` has child component `B`, which has attributes `attr3`, `attr4`, `attr5`.

```
A(attr1, attr2)
          ^
          |
 B(attr3, attr4, attr5)
```

The purpose of rewriting the query in the previous example is to return item type `A` based on conditions in child component `B`.

Correct rewrite

Query `Q1` can be rewritten to make it more efficient. Query `Q2` is the rewritten query. The two queries are equivalent.

```
Q1: /A[B/@attr3 = "a" OR B/@attr4 = "b"]
Q2: /A[B[@attr3 = "a" OR @attr4 = "b"]]
```

In IBM Content Manager, each component is stored in a relational table. Every time you run a query on a component, the predicates are evaluated and a join is performed on that component table. Query `Q1` requires one table reference to item type `A` and two table references to item type `B`. However, because query `Q2` has only one path expression for item type `B` that contains the predicates, it has one table reference to item type `A` and only one table reference to item type `B`.

Table 34. Query table references

Query	Table references in FROM
Q1	A, B, B
Q2	A, B

Because query Q2 involves fewer table references in the generated SQL, it is shorter and evaluates faster that query Q1.

Valid example combining AND and OR

Queries that combine OR and AND operators can be rewritten if you rewrite only the OR part of the query. For example, query Q5 and query Q6 return the same information, but query Q6 is more efficient and requires less memory to run.

```
Q5: /A[(B/@attr3 = "a" OR B/@attr3 = "test1") AND B/@attr5 = "c"]
Q6: /A[B[(@attr4 = "a" OR @attr3 = "test1")] AND B/@attr5 = "c"]
```

Understanding text search

The IBM 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 by 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. IBM 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.

Review the information about DB2 Net Search Extender used in IBM Content Manager Version 8.2 or higher.

dkOracleTextIndexICM
Class interface that extends `dkTextIndexICM`, and represents an OracleText index.

Important: IBM Content Manager Version 8.4 still supports Version 8 functionality such as the definition class `DKTextIndexDefICM` and the methods by using `DKTextIndexDefICM` of `DKItemTypeDefICM` and `DKAttrDefICM`. However, the old functionality is not supported with OracleText.

"Searching for object contents" on page 212
"Searching for documents" on page 212
"Making user-defined attributes text searchable" on page 212
"Understanding text search syntax" on page 212

Related reference

DB2 Net Search Extender Help Overview

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` classes.

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 the `dkOracleTextIndexICM` interface. For more information on the APIs, see the *Application Programming Reference* or the `SItemTypeCreationICM` sample.

Understanding text search syntax

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

“Basic text search”

“Advanced text search” on page 213

“Text search by using UDF ICMFetchFilter” on page 214

“ICMFetchFilter timeout duration” on page 214

“Indexing and searching in a Unicode DB2 UDB library server database” on page 215

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. Basic search is also supported on Oracle library servers.

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 basic text search syntax is as follows:

- 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 about 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, OracleText for Oracle, and IBM Text Search for DB2 for z/OS), so the search syntax will be different. For Oracle, advanced search uses the search syntax supported by OracleText.

`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 IBM 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 IBM Content Manager Version 8 Release 2, the advanced text search functions "contains-text" and "score" are still supported. These functions are identical to contains-text-db2 and score-db2 functions.

`contains-text-db2ts` and `score-db2ts`

Uses DB2TS text search syntax except when handling terms without a plus or a minus sign. Although DB2TS does not support fuzzy searches, you can remove the quotation marks on a term to imply a stemmed search.

Example:

```
/Book[contains-text-db2ts(@Title, "ibm")=1] SORTBY(score-
db2ts(@Title, "ibm"))
```

Also, DB2TS supports additional search argument options: QUERYLANGUAGE, RESULTLIMIT, and SYNONYM.

Example:

```
/Book[contains-text-db2ts(@Title, "ibm", "QUERYLANGUAGE=en_US
RESULTLIMIT=10 SYNONYM=ON")=1]
```

`contains-text-oracle` and `score-oracle`

Uses OracleText search syntax. Performs the same quote translation as `contains-text-db2`. For more information about supported syntax see OracleText documentation.

Example:

`/Book[contains-text-oracle(@Title, "ibm",1) > 0] SORTBY(score-oracle(1))` The `contains-text-oracle` function contains the optional third parameter: label name, which is used as the parameter to `score-oracle`.

For more information about advanced text search, see the `SSearchICM` sample.

Text search by using UDF ICMFetchFilter

The user exit routine, `ICMFetchContentExit (ICMFetchContentParms1 *IFP)` is available to enhance text search.

ICMFetchFilter timeout duration

If you enable and set the timeout duration, the DB2 Net Search Extender (NSE) update process does not get blocked when the INSO conversion stops.

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 the INSO conversion stops.

IBM 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, IBM 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 UDF timeout also does not apply to Oracle.

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*.

Related information

Text search update

Indexing and searching in a Unicode DB2 UDB library server database

In previous releases, users have experienced problems with the way IBM Content Manager handled text indexing of non-English document content when using a Unicode DB2 UDB library server database.

The ICMFetchFilter UDF is used to extract text from non-text documents, such as Word or PDF. In previous releases, this text was returned in the operating system code page, rather than the database code page. Because the text must exist in the database code page for it to be interpreted properly by the DB2 Net Search Extender during indexing, if the code page does not match, only words that exist in both code pages are interpreted, therefore giving the appearance that only English words exist.

You can use a Unicode DB2 UDB library server database to index and search the following types of documents:

- Documents that contain content in multiple languages
- Documents that are in different languages, but belong to the same item type, by using a single UTF-8 text index

In previous versions, you might have configured the **`ICMCCSID`** environment variable to be set to 1208 in the library server environment so that the operating system code page is not required to be the same as the database code page, therefore allowing text indexing of multi-language documents for a Unicode database. However, beginning with version 8.4.3, this setting is no longer necessary and the **ICMCCSID** environment variable is deprecated. Although this environment variable is deprecated, you can allow it to remain at the value that is currently set.

"Setting up your environment for text indexing"

"ManuallyDropDb2CMTextindexes.SQL script" on page 218

Setting up your environment for text indexing:

In order for text indexing of document content to work properly, and before you can search for words in multiple languages, your environment must be set up correctly.

Important: To ensure that plain text documents are indexed properly, you must convert any plain text documents that are in the local code page to UTF-8.

Before you proceed, install IBM Content Manager 8.3 Fix Pack 2 or later and ensure that your library server database is Unicode (code page 1208). When you install IBM Content Manager, you are given the option to select a Unicode database. You can verify that Unicode is installed by connecting to the database and issuing the following command: `db2 get db config`. Entering this command returns the entire database configuration; find the line that contains `Database code page` and verify that the value is 1208.

In previous versions of IBM Content Manager, you must set **CCSID** and the **ICMCCSID** environment variable to 1208 in the library server. However, beginning with Version 8.4.3, the **ICMCCSID** environment variable is deprecated and therefore **CCSID** and **ICMCCSID** no longer must be set. Although the **ICMCCSID** environment variable is deprecated, its existing value does not need to be changed.

The following information contains the main tasks. For Version 8.4.2 and lower, all three steps are required; for Version 8.4.3 and higher, only step 3 is required:

- For Version 8.4.2 and lower:
  - Set the **CCSID** value to 1208 in the library server.
  - Re-create the text indexes.
- For all versions: Ensure that text documents are UTF-8 encoded.

1. *For Version 8.4.2 and lower*: Set the **CCSID** value 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 value of **CCSID** set to 1208. In addition, the **ICMCCSID** environment variable must be set to 1208 in the library server environment. To accomplish these tasks, 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 that the **ICMCCSID** variable is added to DB2ENVLIST:

```
DB2LIBPATH=/usr/lpp/icm/lib
DB2ENVLIST='LIBPATH ICMROOT ICMDLL ICMCOMP
 CMCOMMON DB2LIBPATH ICMCCSID'
```

c. Recycle the DB2 UDB instance by entering

```
db2stop force
db2start
```

Windows
Set the **ICMCCSID** as a system environment variable, then stop and restart DB2 UDB.

2. *For Version 8.4.2 and lower*: Re-create the text indexes. If you have item types with text indexes that 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 end of this topic. You must customize the script to your environment before it can be used to re-create text indexes. Important: This procedure directly modifies the library server database. Therefore, it is recommended that you back up your database and text indexes before you attempt to manually drop and re-create the text indexes.
3. *For all versions*: Ensure that text documents are UTF-8 encoded. Manually indicate that the documents are UTF-8 encoded. If you do not, although text search functions correctly, you 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.

Important: To ensure that plain text documents are indexed properly, you must convert any plain text documents that are in the local code page to UTF-8.

When you have completed all of these tasks, document content can be successfully text-indexed. The text is retrieved in Unicode and then converted to the correct format that is required by DB2 Net Search Extender. Because all of the text is processed in Unicode, text indexing of documents that contain text from multiple languages is enabled.

For new text indexes, search can be performed on words of different languages, subject to the limitations described in the next paragraph. For existing text indexes, the index must be dropped and re-created as described earlier. If plain text documents are added to the same node type, no code page conversion occurs. 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 non-Unicode databases, 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 IBM Content Manager.

Text indexing and search of multiple languages for a Unicode database is supported by using `ICMFetchFilter`. It is also supported when a text attribute is made text searchable (CHAR, VARCHAR, CLOB), but is not supported for an attribute text index that uses `ICMFilter` (BLOB attributes). The IBM Content Manager Java API fully supports searching for words in multiple languages. Because Java represents strings in Unicode, the Unicode characters in a search string in Java flow to the library server correctly.

The C++ API does not support full Unicode, although it can be used to access a Unicode library server. Because the C++ API uses a single, non-Unicode code page, it can search only by using words that can be represented in the code page.

For a list of formats supported by Oracle Outside In Technology, see the following: Oracle Outside In Technology: Supported Formats.

The issue addressed applies only to IBM Content Manager systems that use DB2 UDB. This fix does not apply when running the library server on Oracle. General text indexing and search of data in containing multiple languages is partially supported when running the IBM Content Manager library server on an Oracle database. OracleText 9i MULTI-LEXER can be used with resource item types and custom applications to enable multiple language indexing. OracleText 9i BASIC-LEXER works for a combination of whitespace-delimited languages. 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:

This sample SQL script demonstrates how to remove item type text indexes.

If you want to drop bad text indexes from DB2 Net Search Extender (NSE) and you plan to rebuild a new index for the same item type, only steps 1 and 2 are required to be run before you rebuild the text indexes. However, if you want to remove the text indexes completely from both NSE and IBM Content Manager, all six steps must be performed successfully to ensure complete data removal.

```
-- This sample script demonstrates how to manually drop an item type text index
-- from CM 8. However, once the text index is dropped, only document item types
-- can re-create it. For resource item types, if the text index is dropped it
-- cannot be re-created by using the admin client.
--
-- This procedure directly modifies the library server database. Therefore, it
-- is recommended that you back up your database and text indexes before you
-- attempt to manually drop and re-create the text indexes.
--
-- Before running steps 2, 3 and 5, review the output of the queries
-- from earlier steps and then customize the script.
--
-- Connect to the library server database.
-- Change the following command with your database name, user ID, 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 following example finds the index names for the item type 'Forum'.
-- To find all item type indexes in the system, remove the last AND condition
-- from the sample query so that it will show text indexes for all item
-- types. Note that item type index names end with 'TIE' and that 'Forum'
-- in the last predicate of "AND a.keywordname = 'Forum'" is an example of an
-- item type name. You must replace 'Forum' with your own item type name. If a
-- schema other than 'icmadmin' is used, it must be substituted in the
-- following example:
select a.keywordname AS CompTypeName, d.indexname AS IndexName, b.itemtypeid
      from icmadmin.icmstnlskeywords     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.
-- Note that ICMUU01033001TIE is a sample index name; you must use the index
-- name obtained from the query in step 1.
-- As in the connect example, customize the database name, user ID, and
-- password in the following command:
db2text drop index icmadmin.ICMUU01033001TIE for text connect to ICMNLSDB
   user icmadmin using password

-- Step 3:
-- Delete the row from the CM table that holds the configuration of the
-- text index. This step is required to allow the index to be re-created with
-- a new configuration. Note that this step deletes all configuration for
-- this index. You must re-enter the configuration information when you
-- re-create the index. It is suggested that you review the index configuration
-- in the System Admin GUI before performing this step. Replace the indexname
-- 'ICMUU01033001TIE' in the following delete command with the index name
-- obtained from the from query in step 1:
DELETE FROM icmadmin.icmsttextindexconf WHERE indexname='ICMUU01033001TIE';

-- Step 4:
-- In step 5, you must inform CM that this item type is no longer searchable by
-- setting a flag in the component type definition of the attribute that has
-- the text index. For item type text indexes, the attribute is named TIEREF.
-- To prepare for step 5, determine the current value of the TIEREF flag
-- by running a query. In this query, set the component type that contains the
-- index and the name of the attribute that contains the text index.
-- For resource item types, this name is the name of the item type. For
-- document item types, this name is derived from the ItemTypeId that was
-- returned from the query in step 1. The name of the comp type for a document
-- item type is ICMPart + the item type ID. In the following query example,
-- this name is 'ICMParts1009':
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 out the following line:
                      a.keywordname      = 'ICMParts1009' AND
                      kw.keywordclass    = 1 AND
                      kw.keywordcode     = ca.AttributeId AND
                      kw.keywordname     = 'TIEREF';;

-- Step 5:
-- Inform CM that this item type is no longer text searchable by setting the
-- attribute flags (ATTRFLAGS) for the TIEREF attribute to 3.
-- Customize the following query to replace componenttypeid in the predicate
-- "ca.componenttypeid = 1033" with the component type id obtained from the
-- query in step 4:
UPDATE icmadmin.ICMSTCompAttrs ca
             SET    ca.ATTRFLAGS       = 3
             WHERE  ca.componenttypeid = 1033 AND
                    ca.attributeid     = 20   AND
                    ca.attributegroup  = 7
;

-- Step 6:
-- Commit all the changes:
```

```
COMMIT WORK;

-- Disconnect from the database:
CONNECT RESET;

-- After these steps are done, you can use the System Admin GUI to re-create
-- the text index for the 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 by using both parametric and text search.

"Creating combined parametric and text search: Java"

"Creating combined parametric and text search: C++" on page 221

Creating combined parametric and text search: Java

Combining parametric and text searches in Java involves five 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 five 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())
     ->getItemId() << "  (" <<     ((DKPidICM*)ddo->getPidObject())
     ->getObjectType() << ")" << 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 by 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 the following figure. See the data model illustration when you review the sample queries.

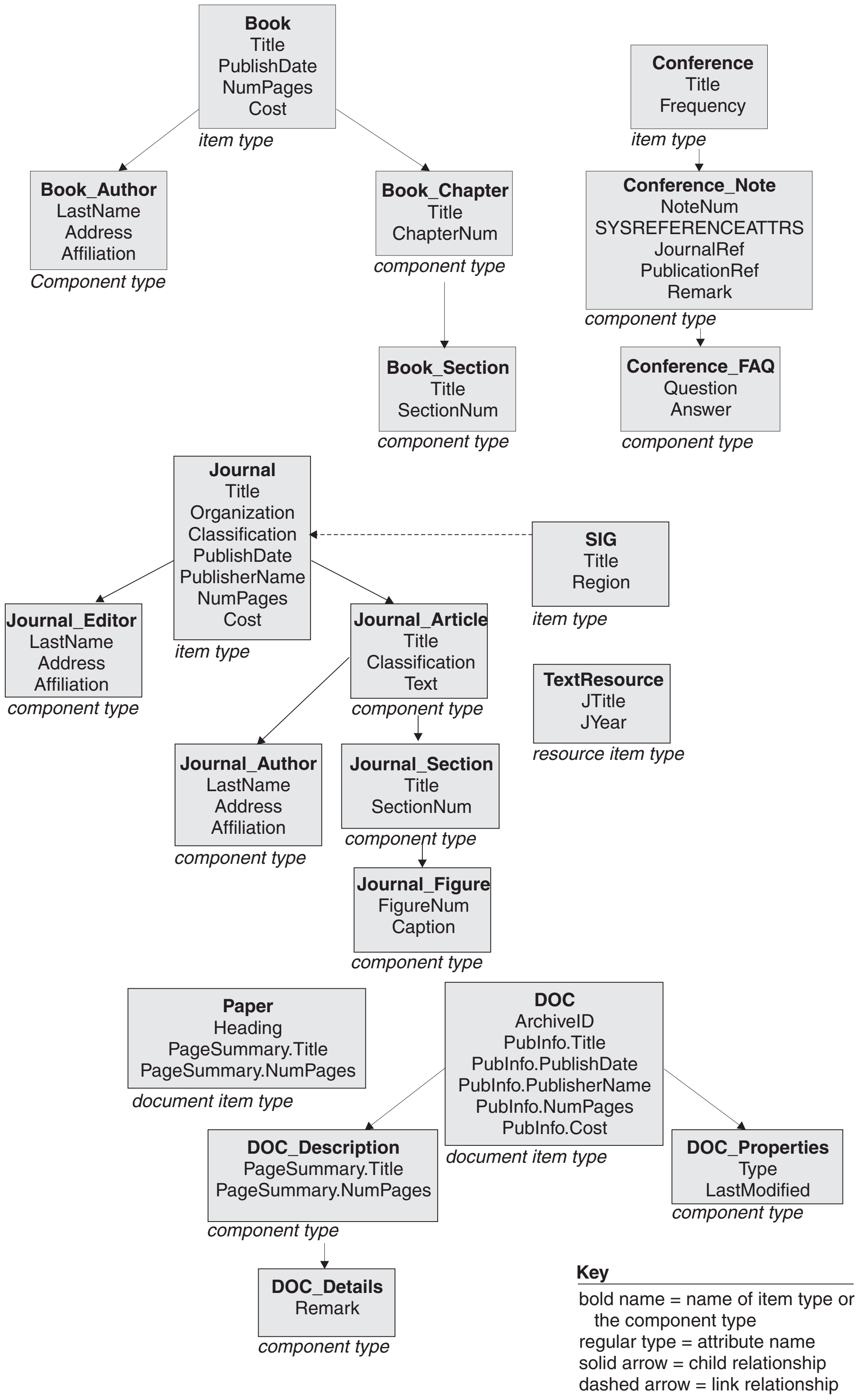


Figure 9. Query examples data model

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 the previous figure.

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>)

        <ICMCHECKEDOUT (STRING ICMCHKOUTUSER, TIMESTAMP ICMCHKOUTTS)
        </ICMCHECKEDOUT>
    ...(repeating <ICMCHECKEDOUT>)

</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 correct 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 by 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 by using a set operator (UNION).

Examples of incorrect usage

Examples of invalid usage of INTERSECT/EXCEPT:

/Journal[(Journal_Editor/@LastName INTERSECT .//Journal_Author/@LastName) = "Davis"]
: This query is incorrect because the result of INTERSECT operator is combined by using a comparison operator (=).

/Journal[(.//Journal_Section/@SectionNum EXCEPT .//Journal_Figure/@FigureNum) + 5 = 10]
: This query is incorrect because the result of EXCEPT is combined by using an arithmetic operator (+).

/Doc[@ArchiveID = 555] INTERSECT /ICMBASE

This query is incorrect because the INTERSECT keyword is appropriate when there is actually overlap between the results. In this example, there is no item of type Doc that is the same as an item of type ICMBASE.

To properly return all parts of type ICMBASE belonging to a document of type Doc with ArchiveID 555, use the following query:

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

Important: If you already have a DDO containing the Doc item with ID of 555, you can use the retrieve method to retrieve the parts that belongs to this document.

"Query examples"

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 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 about 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 9 on page 223, and the sample XML document.

Using the IN operator

The IN operator is used with literals (integer, float, string), and should be used to match an attribute with a set of possible values. Using IN instead of multiple equality or inequality predicates connected through OR operators result in shorter queries. The IN operator should be used if existing queries cause a `DK_ICM_MSG_QL_TOO_LONG_OR_TOO_COMPLEX` exception.

To search for a journal that has 4, 8, or 12 pages, the search string without the IN operator:

```
/Journal[(@NumPages = 4) OR (@NumPages = 8)
OR (@NumPages = 12)]
```

To search for a journal that has 4, 8, or 12 pages, the search string with IN operator is:

```
/Journal[@NumPages IN (4,8,12)]
```

To search for books written by authors with last names other than Smith or Jones, the search string without the IN operator:

```
/Book[Book_Author[(@LastName != "Smith")
AND (@LastName != "Jones")]]
```

To search for books written by authors with last names other than Smith or Jones, the search string with the IN operator:

```
/Book[Book_Author[@LastName NOT IN ("Smith", "Jones")]]
```

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`).

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`.

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 `%`.

BETWEEN operator: 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.

If you are looking for all journals with the number of pages between 10 and 200, you can use the BETWEEN operator available in the query language.

```
"/Journal [@NumPages BETWEEN 10 AND 200]"
```

The values used with the BETWEEN operator are included in the range, meaning that the lower bound includes the value of 10 and the upper bound includes the value of 200.

The use of the BETWEEN operator should make your query more easy to read, especially if multiple query conditions are used in the same query string. The BETWEEN operator can be used not only with attributes of numeric types, but also with attributes of other types (for example, date, string).

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.

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. The following examples perform text searches with the `contains-text` function. For the syntax supported by this function, see the 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`.

Text search for DB2

```
//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text-db2(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

Text search for Oracle

```
//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text-oracle(@Text, " 'Java' & 'XML' ", 1)>0]
SORTBY(score(1))
```

Advanced text search on z/OS (contains-text-db2ts and score-db2ts functions)

In IBM Text Search for DB2 for z/OS, you can append additional search argument options to the query:

QUERYLANGUAGE
Specifies the locale of the query. The default is en_US.

RESULTLIMIT
Specifies the maximum number of results to return from the underlying search engine. The value can be an integer value between 1 and 2 147 483 647. If the RESULTLIMIT option is not specified, then the query does not limit the results.

SYNONYM
Specifies whether to use a synonym dictionary that is associated with the text search index. You can add a synonym dictionary to a collection by using the synonym tool. The default is OFF.

For more information on IBM Text Search for DB2 for z/OS, see the DB2 for z/OS Information Center.

The following query finds ten students at random who wrote online essays that contain the words "fossil fuel" in Spanish.

Text search for DB2TS

```
//STUDENT_ESSAYS[contains-text-db2ts(@TERM_PAPER, "combustible fosil",
  "QUERYLANGUAGE=es_ES RESULTLIMIT=10 SYNONYM=ON")=1]
```

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 by 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`.

Text search for DB2

```
/Journal
[Journal_Article[contains-text-db2(@Title, " 'Design' |
  'Index'  ")=1]]
SORTBY (@Title DESCENDING)
```

Text search for Oracle

```
/Journal
[Journal_Article[contains-text-oracle(@Title, " 'Design' |
  'Index'  ")>0]]
SORTBY (@Title DESCENDING)
```

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` by 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 by 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.

The `score-basic` and the `score` functions support the SORTBY clause. However, the SORTBY clause can be used in many functions provided that they return a scalar result that can be used to sort the result. For example, the following query is valid for returning all versions of all journal items in the system sorted by length of the title attribute value:

```
/Journal SORTBY( LENGTH(@title) )
```

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.

Text search for DB2

```
/TextResource[contains-text-db2(@TIEREF, " 'Java' & 'XML'
")=1]
```

Text search for Oracle

```
/TextResource[contains-text-oracle(@TIEREF, " 'Java' & 'XML'
")>0]
```

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
```

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
```

Traversal of references in the reverse direction

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

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

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.

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%"]
```

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(.)]
```

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(.)]
```

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(.)]
```

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.

Text search for DB2

```
/Doc[contains-text-db2(.//ICMPARTS/@TIEREF, " 'XML' ")=1]
```

Text search for Oracle

```
/Doc[contains-text-oracle(.//ICMPARTS/@TIEREF, " 'XML' ")>0]
```

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 => *
```

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.

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]
```

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"]]
```

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 by 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.

List of a result of query

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

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

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`.

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.

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 by using upper case. All attribute data pertaining to user IDs must store them in upper case as well.

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"]
```

Related reference

DB2 Net Search Extender Help Overview

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 by 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”

“Using escape sequences with the LIKE operator”

“Using escape sequences with advanced text search” on page 237

“Using escape sequences with basic text search (contains-text-basic and score-basic functions)” on page 238

“Using escape sequences in Java and C++” on page 240

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 quotation mark with another double quotation mark.

```
//Journal_Article[@Title = "Analysis of ""The Time Machine"" by H. G. Wells
himself"]
```

Because the article's title contains the name of the book in double quotation marks, `The Time Machine`, these internal double quotation marks need to be escaped.

Example: single quotation mark (apostrophe) '

You do not need to escape the quotation marks 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 wildcard characters with attributes of character and variable character types”

Using wildcard characters with attributes of character and variable character types

When you use wildcard characters with the LIKE operator in IBM Content Manager queries, the queries on the attributes of character types behave differently than the queries on the attributes of variable character types. The difference in behavior occurs because the length of the attributes of character types are fixed. To make the queries more effective, you can use the multiple wildcard characters to represent all trailing characters after the search term.

For example, if in the `Journal` component type view, the `PublisherName` attribute is defined to be a variable character type and one of the components has the value IBM in this attribute, then the following query returns the result that contains this attribute value because the `PublisherName` attribute is of variable character type. If the value `IBM` is stored in this attribute, the length of the attribute is 3 and the following query returns the results that contain the string `IBM`.

```
/Journal [@PublisherName LIKE "IB_"]
```

However, if in the `Journal_Editor` component type , the `Affiliation` attribute is defined to be a character type and one of the components has the same value `IBM` in this attribute, then the following similar query does not return the result that contains the value.

```
//Journal_Editor [@Affiliation LIKE "IB_"]
```

When the value IBM is stored in the attribute of fixed character type, the three characters of the word IBM are followed by 61 blank spaces (64 - 3 = 61) because the `Affiliation` attribute contains 64 characters. These blank spaces are padding characters, where the exact type of character used depends on the database settings. Because the wildcard underscore character (_) that is used with the LIKE operator matches any single character, only values starting with two letters I and B, followed by any third letter are returned in the example query on the `Affiliation` attribute in the previous section. Because all values in the `Affiliation` attribute contain 64 characters (not 3 characters), the following query does not return any results:

```
//Journal_Editor [@Affiliation LIKE "IB_"]
```

To make the queries less error-prone because of this difference, you can use the multi-character wildcard to represent all trailing characters after the search term. To find all results with attribute values that start with a specific pattern for attributes of fixed character length, you can use the multiple wildcard characters % as follows:

```
//Journal_Editor [@Affiliation LIKE "IB%"]
```

Remember: Using the wildcard characters requires extra processing on the underlying database and can lead to a degradation of performance for your queries. To get better performance in your queries, use the exact phrase that you want to search for (without the LIKE operator or wildcard characters). For example:

```
//Journal_Editor [@Affiliation = "IBM"]
```

Using escape sequences with advanced text search

Escape sequences are necessary when quotation marks and wildcard symbols are used in text searches.

Double quotation mark

Precede a double quotation mark with another double quotation mark.

DB2

```
//Journal_Article[contains-text-db2 (@Title,
    " 'Analysis of ""The Time Machine"" %' ")=1]
```

Oracle

```
//Journal_Article[contains-text-oracle (@Title,
 " 'Analysis of ""The Time Machine"" 0]
```

Because the title of the article contains the name of the book in double quotation marks, "The Time Machine", these internal double quotation marks must 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, the apostrophe must 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' "))
```

Oracle

```
/Book[contains-text-oracle (@Title,
" 'Uncle Tom''s Cabin' ")>0]SORTBY
 (score-oracle (@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' "))
```

Oracle

```
/Book[contains-text-oracle (@Title, " ('Greek' & 'Plato''s Symposium')
 & NOT ' Socrates' ")=1] SORTBY
(score-oracle (@Title, " ('Greek' & 'Plato''s Symposium')
                & NOT ' Socrates' "))
```

Note: `Plato''s` has two apostrophes.

Wildcards ("%", "_")

Just as the LIKE operator, advanced syntax uses the percent (%) and underscore (_) symbols as wildcard symbols. The percent symbol (%) is a wildcard symbol that is used to represent any number of arbitrary characters. The underscore symbol (_) is a wildcard symbol used to represent a single arbitrary character. If you want a wildcard symbol to be treated as a regular character, follow these instructions:

1. Precede the wildcard character with an escape character.
2. Add an escape clause after each term in which 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 symbol.

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]
```

An escape clause must be added after every term in your text search string in which you escape wildcard symbols, 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. Improper use of 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 IBM 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” on page 241

“Sample usage scenario” on page 242

“Description of filtering behavior” on page 243

“Performance considerations” on page 245

"Database indexes for each filtered attribute" on page 246
"Security implications" on page 247

Important considerations for using row-based filtering

If you use row-based filtering, there are some important usage guidelines to 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 IBM 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 subset, then IBM 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 by using the Client for Windows or eClient, you open a folder or worklist.
- When you create a custom application (either Java or C++) by using the IBM Content Manager API that calls the `getWorkList` function or that retrieves a folder item by using `DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND` or `DKConstant.DK_CM_CONTENT_ITEMTREE` (in IBM Content Manager Version 8.2 and all fix packs, and IBM Content Manager Version 8.3, fix pack levels earlier than Fix Pack 4) or the method `DKRetrieveOptionsICM::linksOutbound(true)` (in IBM Content Manager Version 8.3 Fix Pack 4 and later).

In general, these problems can happen because of two main causes:

- 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 IBM 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 might be a wrong view that has a row-based filter condition, a condition that filters out the expected documents.
- 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 many IDs, then the performance of the retrieve operation might be affected.

If you use row-based filters on subsets, be aware of how these subsets can affect the result sets as previously described. Carefully design your subsets, filter conditions, and ACLs accordingly.

Tip: Do not use a row-based filter condition if you already use auto-linking 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. Performance problems occur 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 IBM 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.

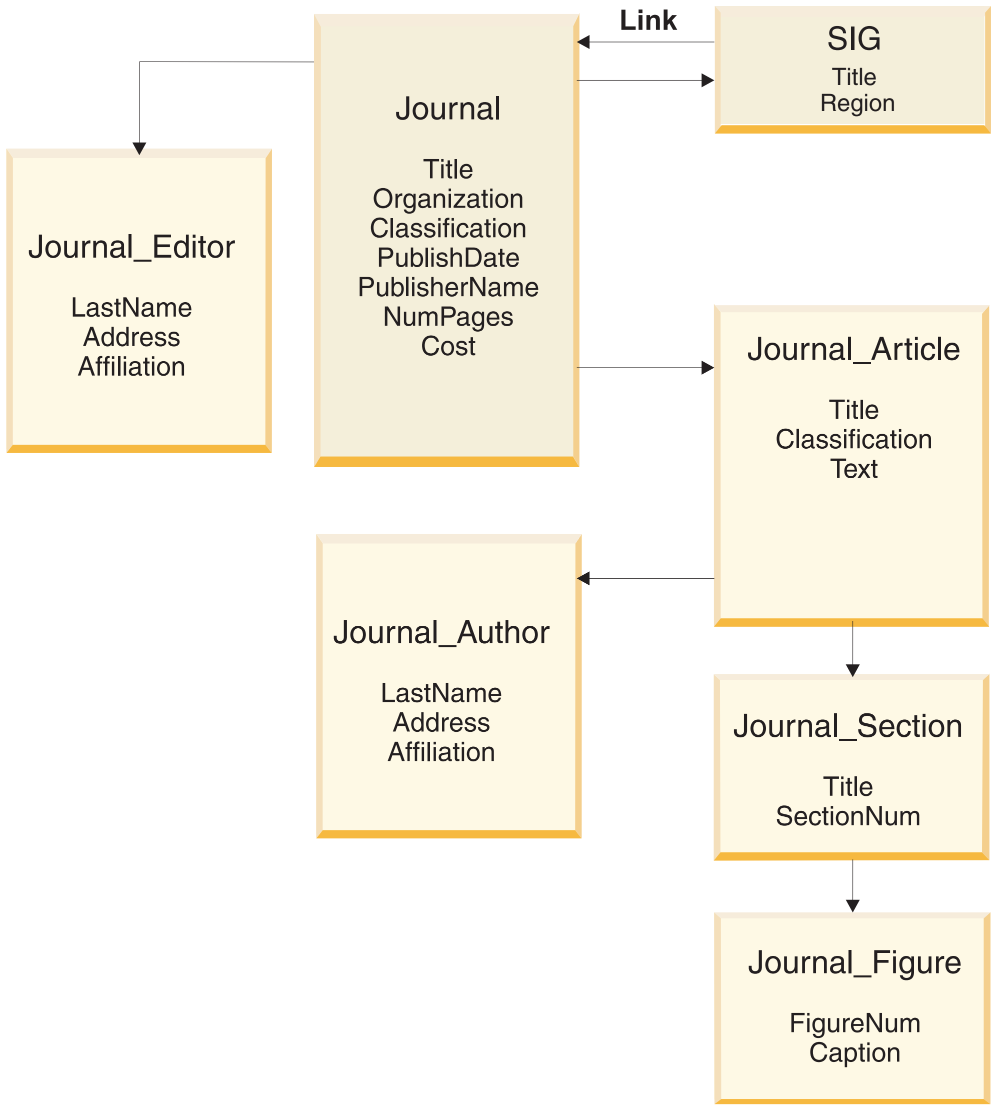


Figure 10. Sample data model for query row-based view filtering

Table 35. Sample definition of views with filters

User component type view	Base component type	Filter condition
MyJournal	Journal	@Organization = "IBM"
MyJournal_Article	Journal_Article	@Classification = "Public"
MyJournal_Section	Journal_Section	None
MyJournal_Figure	Journal_Figure	@FigureNum = 5
MyJournal_Author	Journal_Author	@Affiliation = "Almaden"
MyJournal_Editor	Journal_Editor	@Affiliation = "Almaden"
MySIG	SIG	@Region = "USA"

Description of filtering behavior

How row-based view filtering behaves depends on the type of query that you want to perform. Row-based filtering in IBM 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 MyJournal_Editor and MyJournal_Figure filters are not applied for the "/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 IBM 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 by 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 IBM 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 helps 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 IBM 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 an IBM 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 IBM 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 ATTR*XXXXX*00110 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 by 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 by 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 quotation marks and can contain any character except a double quotation mark. To include a double quotation mark as the part of the string literal, specify two consecutive double quotation marks; that is, a double quotation mark escaped by another double quotation mark. These double quotation marks are treated as a single double quotation mark *)
- **STRING_LITERAL**= '"', { (**UNICODE_CHARACTER** - '"'), | ('"', '"')}, '"';
- (* Escape sequence is a single character delimited by double quotation marks. To specify a double quotation mark as the escape character, specify two consecutive double quotation marks; that is, a double quotation mark escaped by another double quotation mark. These double quotation marks are treated as a single double quotation mark. *)
- **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 quotation marks. *)
- **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 ;**

Related reference

LIKE predicate

Routing a document through a process

The document routing APIs enable you to build new applications by using document routing, or add document routing functionality into your existing applications.

IBM 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 IBM Content Manager transactions.
- Presentation of the work that only the user can access.
- Single audit trail that includes records for document creation, modification, and routing.

Review the basic document routing concepts and terminology. Also refer to the samples for additional document routing information.

“Understanding the document routing process”

“Document routing classes” on page 255

Related reference

Managing document routing with IBM Content Manager

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 by using the system administration client.

The basic operations you can perform by using document routing include:

- Starting a process
- Ending a process
- Continuing a process
- Suspending a process
- Resuming a process
- Getting work from a worklist
- Getting the next item from a worklist
- Defining, updating, and deleting a process
- Defining, updating, and deleting work node
- Defining, updating, and deleting a worklist

To define document routing processes in IBM Content Manager, you use the graphical workflow builder provided by IBM Content Manager to model your document flows of business processes. Before a document routing process definition can be used to route documents, it must be verified by the graphical workflow builder. The graphical builder imports and exports 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.

"Action"

"Workflow node variables"

"Adding and updating a document routing process" on page 253

"Understanding compatibility with versions prior to Version 8.3" on page 254

"Changing the DOCROUTINGFREQ column in the ICMSTSYSCONTROL table for DB2" on page 254

"Changing the DOCROUTINGFREQ column in the ICMSTSYSCONTROL table for Oracle" on page 254

Related reference

Modeling the process graphically

Modeling the workflow graphically

Parallel routes

Decision point

Creating parallel routes

Business application

Action

Action list

Process

Action

Using workflow action objects, you can customize workflow applications to integrate with external systems.

For example, the workflow action object can refer to an external DLL or Java™ class to retrieve and email a document. The application integrator is responsible for invoking, validating, and implementing the external DLL or Java class described by the workflow action. IBM Content Manager never acts on any of the workflow actions in the objects. IBM Content Manager merely holds the workflow actions in the objects for a separate application to read.

Related reference

Action

Workflow node variables

Workflow node variables represent instances of name-value pairs (strings) within work packages.

The workflow node variables serve as helper objects that can describe data types and expected run time 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 node variables during run time. Note that IBM Content Manager does not enforce any semantic checking or validation for the name-value pairs against their corresponding container variable definitions.

Related reference

Modeling the process graphically

Modeling the workflow graphically

Parallel routes

Decision point

Creating parallel routes

Business application

Action

Action list

Process

Related information

Understanding compatibility with versions prior to Version 8.3

Document routing classes

Document routing constants

Programming document routing user exits

Adding and updating a document routing process

With the graphical workflow builder, you draw the workflow process using nodes and connectors. The `add(DKProcessICM process)` and `update(DKProcessICM process)` methods can save only the processes that passed the graphic workflow builder verification.

The add and update methods work only 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 the following table for all possible combinations.

Table 36. Combinations that the add(DKProcessICM process) and update(DKProcessICM process) method can save

Process state	Diagram definition	Route definition	Can be saved
Draft (0)	No	No	No
Draft (0)	No	Yes	No
Draft (0)	Yes	No	Yes
Draft (0)	Yes	Yes	No
Verified (1)	No	No	No
Verified (1)	No	Yes	Yes
Verified (1)	Yes	No	No

Table 36. Combinations that the add(DKProcessICM process) and update(DKProcessICM process) method can save (continued)

Process state	Diagram definition	Route definition	Can be saved
Verified (1)	Yes	Yes	Yes

Related reference

Modeling the workflow graphically

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 by 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`* `by 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`

Changing the DOCROUTINGFREQ column in the ICMSTSYSCONTROL table for Oracle

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 defining the document routing frequency checking value on Oracle, complete the following steps:

1. At the sqlplus command prompt, enter: `connect userID/password @ dbname`
2. Determine the current DOCROUTINGFREQ setting. Enter the following:`select docroutingfreq from icmstsyscontrol`

   The default setting is 10, which means the minimum time the work package notify and suspend flags are checked is 10 minutes.
3. Specify the document routing frequency value.

   ```
   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:`quit`

Document routing classes

IBM Content Manager provides 12 classes that you can use to implement document routing functionality into your application.

You can find details about these classes and methods in the *Application Programming Reference*. The document routing classes and methods 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.

Although it is possible to create Version 8.4 document routing process definitions with the API directly, 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 avoids this behaviour 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, use the graphical workflow builder to handle this status 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 run time.

The listProcessNames() and listProcesses() methods in DKDocRoutingServiceMgmtICM list only verified processes. Processes in draft state are not returned.

The add(DKProcessICM process) and update(DKProcessICM process) methods can save only the processes that passed the graphic workflow builder verification. Therefore, the add and update methods work only 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 regulates 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 the following table for all possible combinations.

Table 37. Combinations that the add(DKProcessICM process) and update(DKProcessICM process) method can save

Process state	Diagram definition	Route definition	Can be saved
Draft (0)	No	No	No
Draft (0)	No	Yes	No
Draft (0)	Yes	No	Yes
Draft (0)	Yes	Yes	No
Verified (1)	No	No	No
Verified (1)	No	Yes	Yes
Verified (1)	Yes	No	No
Verified (1)	Yes	Yes	Yes

DKWorkNodeICM

This class represents a work node definition which details the expected or applicable tasks (such as ActionList and library server exit program) to perform at that particular step in a process.

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 run time behavior (such as name,"prompt, and display to user) for workflow container data routed along a workpackage.

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. IBM 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. IBM Content Manager does not automatically start the exit routine that you specify in the `DKWorkFlowActionICM` object. A custom application must start, validate, and implement the action.

IBM Content Manager pre-defines several actions. They primarily help IBM clients (such as eClient and pClient) display available menu selections at a worknode. IBM Content Manager does not automate or enforce any of these system-defined workflow actions.

DKWorkFlowActionListICM

This class records a collection of actions and its name is returned along with the retrieval of a worknode. This function enables client application to display applicable 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 about 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” on page 259
“Defining a new regular work node” on page 259
“Listing work nodes” on page 261
“Defining a new collection point” on page 262
“Defining a worklist” on page 264
“Listing worklists” on page 266
“Defining a new process and associated route” on page 266
“Starting a document routing process” on page 269
“Ending a process” on page 270
“Continuing a process” on page 270
“Suspending a process” on page 271
“Resuming a process” on page 272
“Listing work package persistent identifier strings in a worklist” on page 272
“Retrieving work package information” on page 273
“Listing document routing processes” on page 274
“Implementing ad hoc routing” on page 275
“Document routing example queries” on page 276
“Granting privileges for document routing” on page 277
“Working with access control lists for document routing” on page 278
“Programming document routing user exits” on page 278

"Document routing constants" on page 279

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. For example `start`, `terminate`, `continue`, `suspend`, `resume`, `listWorkPackages`, and `setWorkPackageContainerData`.

The `DKDocRoutingServiceMgmtICM` object is a helper class that provides methods to manage the document routing definition classes: `DKProcessICM`, `DKWorkNodeICM`, `DKWorkFlowActionICM`, `DKWorkFlowActionListICM` and `DKWorkListICM`, and the metadata required to define the objects.

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.

The following examples 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. The `DKWorkNodeICM` class represents a work node definition which details the expected or applicable tasks (such as action list and library server exit program) to perform at that particular step in a process.

It might 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.

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.

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 routine 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 routine 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.

Related reference

Defining work nodes outside of the graphical process builder

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 about listing work nodes, refer to the `SDocRoutingListingICM` sample.

Related reference

Defining work nodes outside of the graphical process builder

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 routine 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 routine 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 might 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 about defining collection points, refer to the `SDocRoutingDefinitionCreationICM` sample.

Related reference

Creating a collection point

Defining a worklist

A worklist allows an administrator or a user application to dynamically change work assignments without contacting a user. The `DKWorkListICM` 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.

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 about defining worklists, refer to the `SDocRoutingDefinitionCreationICM` sample.

Related reference

Creating a worklist

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.

Related reference

Creating a worklist

Defining a new process and associated route

The `DKProcessICM` 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 time limit) and description of a process. The DKRouteListEntryICM 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.

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.

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.

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.

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 by 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 might exist between nodes. A specific route might be selected
// by a user-defined "selection" keyword.Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", and so on.
// 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 might
//re-use the same nodes and multiple routes between nodes  might 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 by 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 might exist between nodes.
// A specific route might be selected by a user-defined "selection"
// keyword.Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", and so on .

// 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.

Related reference

Defining a new process

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 following example 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 might 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 might 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.

Related reference

Defining a new process

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 about terminating a process, refer to the `SDocRoutingProcessingICM` sample.

Related reference

Defining a new process

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 following code snippet, the name of the selection that causes the transition from the current work node to the next work node is "Continue". The work package PID string of the current work package is specified in the method call. 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.

Related reference

Defining a new process

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++ run time 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.

Related reference

Defining a new process

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 about resuming a process, refer to the `SDocRoutingProcessingICM` sample.

Related reference

Defining a new process

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 38. listWorkPackagePidStrings API results based on owner

System administration setting for a worklist	owner=empty	owner=logged on user ID	owner = a user ID other than the logged on user ID
Filter on owner is not checked	Returns all work packages in the worklist.	Returns all work packages in the worklist.	Returns all work packages in the work list.
Filter on owner is checked	Returns work packages of the logged on user ID in the worklist.	Returns work packages of the logged on user ID in the worklist.	Returns work packages of the specified user ID in the worklist if the ICM_PRIV_ITEM_GET _ASSIGN_WORK 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

The DKWorkPackageICM 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).

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. When a process starts, one or more work packages are created as a result of the API call.

When a process is started, a work package is created by the library server. Each work package carries a persistent identifier (PID) for a document or folder being routed. However, the library server does not check whether the PID belongs to a document/folder item or a non-document/folder item. Therefore, an application needs to ensure that a process is started only with a document or a folder and is not started with a non-document or non-folder.

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 reference

Defining a new process

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 run time, 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.

Related reference

Ad hoc routing process

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 `AccidentInvesigation` 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 IBM 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 IBM 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 39. Document routing privileges

Privilege	Description	Related API
ICM_PRIV_ITEM _UPDATE_WORK	Used to see if the user is authorized to do the following for a work package: set the priority set the owner set the resume list set the duration for suspension	`suspendProcess` `resumeProcess` `setWorkPackagePriority` `setWorkPackageOwner`
ICM_PRIV_ITEM _ROUTE_START	Used to see if the user is authorized to start a process.	`startProcess`
ICM_PRIV_ITEM _ROUTE_END	Used to see if the user is authorized to terminate a process.	`terminateProcess`

Table 39. Document routing privileges (continued)

Privilege	Description	Related API
ICM_PRIV_ITEM _GET_WORKLIST	Used to see if the user is authorized to get the count or work packages from a worklist.	`getCount` `listWorkPackagePidStrings`
ICM_PRIV_ITEM _GET_WORK	Used to see if the user is authorized to get a work package.	`getNextWorkPackagePidString` `getNextWorkPackage` `checkOutItemInWorkPackage` `retrieveWorkPackage`
ICM_PRIV_ITEM _GET_ASGN_WORK	Used to see if the user is authorized to get a work package that is owned by a different a different user.	`getNextWorkPackagePidString` `getNextWorkPackage` `getCount` `listWorkPackagePidStrings`
ICM_PRIV_ITEM _ROUTE	Used to see if the user is authorized to route a work package.	`continueProcess`

Related reference

Planning user authorization

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 IBM Content Manager document routing entity and their associated privileges are listed in the following table.

Table 40. Access control lists and document routing

Objects	Related methods	Privileges
Process	`startProcess`	ICM_PRIV_ITEM_ROUTE_START
Work node	`continueProcess` `suspendProcess` `resumeProcessterminateProcess` `setWorkPackagePriority` `setWorkPackageOwner`	ICM_PRIV_ITEM_ROUTE ICM_PRIV_ITEM_ROUTE_END ICM_PRIV_ITEM_UPDATE_WORK
Worklist	`getNextWorkPackagePidString` `getNextWorkPackage` `getCount` `listWorkPackagePidStrings` `checkOutItemInWorkPackage`	ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST

Related reference

Planning users' view of the work

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 IBM Content Manager calls a user exit routine, you can retrieve the work package from the work package table by 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 supported only 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 IBM 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 in **`DK_ICM_DR_SELECTION_FILTER_YES`** = 1;
- public final static int **`DK_ICM_DR_SELECTION_FILTER_EITHER`** = 2;
- public final static int **`DK_ICM_DR_SELECTION_ORDER_PRIORITY`** = 0;
- public final static int **`DK_ICM_DR_SELECTION_ORDER_TIME`** = 1;
- public final static int **`DK_ICM_DR_MAX_RESULT_ALL`** = 0;

Workflow node 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

Developing FileNet Business Process Manager workflow applications with IBM Content Manager

The integration of FileNet Business Process Manager with IBM Content Manager helps you develop FileNet Business Process Manager workflow applications with IBM Content Manager.

With FileNet Business Process Manager, you can:

- Integrate with the rules engine to help workflow authors and business analysts to create and add business rules to individual steps of a workflow definition
- Assign values and milestones to a step in the workflow to track the progress of a workflow
- Assign work to individuals and groups of users
- Use tools, such as Business Activity Monitor, Process Tracker, Process Analyzer, and Process Simulation, to analyze the information captured from active processes in the Process Engine

When you design an application in an integrated environment, you can use the IBM Content Manager APIs to access the documents from the IBM Content Manager database and use the FileNet Business Process Manager APIs to start a workflow to route the documents.

To integrate FileNet Business Process Manager with IBM Content Manager, you monitor the events, configure IBM Content Manager in the Component Integrator to use FileNet Business Process Manager, and design a process application by using the `DKContentOperationsICM` APIs and the FileNet Business Process Manager APIs.

1. "Developing process applications with FileNet Business Process Manager"
2. "Monitoring and handling events" on page 283
3. "Configuring IBM Content Manager to use the FileNet Business Process Manager workflow" on page 308
4. "Designing FileNet Business Process Manager workflow applications" on page 312

Related information

Integrating with FileNet Business Process Manager

Example sample configuration

Software requirements

Developing process applications with FileNet Business Process Manager

FileNet Business Process Manager process applications help users to manage information and access resources that are associated with a workflow.

To develop a process application, you must set up your development environment, design the workflow user interfaces, access data and resources, and create interfaces to the Process Engine and related services to perform tasks associated with a step or an operation.

For documentation for developers, in the FileNet P8 Documentation table of contents, click **Developer Help** > **Process Engine Development**.

For information about how to use process applications, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow overview** > **Getting Started**.

"Step processors"

"Work performers"

"User inboxes"

Step processors

A step processor is a process application that performs the operations associated with a step in a workflow.

A step processor processes the information and resources that a workflow user requires to complete a step in a workflow. When a workflow user opens a work item, the step processor shows the instructions, the attachments stored in the object or file stores, current field values, response options, or other resources necessary to allow the user to process the step. When you design or develop a custom step processor, you must provide complete information and data that is required to complete each step in the workflow.

A launch step processor is a specialized type of step processor that begins a workflow. A typical launch step processor contains all of the information that is necessary to initialize a workflow. Because a launch step processor is independent of the condition of workflow progress, it only needs to include the information that the launch step processor introduces into the workflow.

For more information about step processors, in the FileNet P8 Documentation table of contents, click **Developer Help** > **Process Engine Development** > **Developer's Guide** > **Introduction to Process Applications**.

Work performers

A work performer is a process application that performs an operation or set of operations that are associated with a workflow step.

The work performers do not require a user interface. Typical work performer operations include:

- Logging on to a Process Engine and establishing a session
- Polling a work queue to find operations that are related to a workflow step
- Locking the retrieved object
- Processing the work, such as updating and saving data
- Cycling back to queue polling

For documentation for developers, in the FileNet P8 Documentation table of contents, click **Developer Help** > **Process Engine Development** > **Developer's Guide** > **Introduction to Process Applications**.

User inboxes

A user inbox is an application that provides the user, which can be a workflow participant or Tracker, with notifications for work items. It is typically in the form of an HTML page or a Java applet.

The notifications in the user inbox page include the information and associated attachments that are provided from a user or work queue. Some common ways that you can customize a user inbox include building a user inbox UI, displaying a list of queues, querying for and displaying queue contents, and opening a step processor or Tracker assignment. To open a user inbox page, click the **Tasks** tab in the Workplace client of FileNet Business Process Manager.

For documentation for developers, in the FileNet P8 Documentation table of contents, click **Developer Help** > **Process Engine Development** > **Developer's Guide** > **Introduction to Process Applications**.

Monitoring and handling events

IBM Content Manager provides an event monitor and an event handler that enable IBM Content Manager to notify the FileNet Business Process Manager with events that occur in the library server.

When you create an item type, you can enable event subscriptions on the item type by using the system administration client. When an item type is enabled for event subscription, a library server event is generated every time an item of that item type is created, deleted, or updated.

The event monitor does the following:

1. Transforms the library server event to an IBM Content Manager event.
2. Packages the event in the Common Base Event (CBE) format as an XML string with the IBM Content Manager event.
3. Sends the CBE-formatted event to the event handler in form of a Java Message Service (JMS) message.

“Event monitor”

“Event handler” on page 298

“Managing event subscriptions” on page 301

Related information

Example sample configuration

Event monitor

The event monitor is a stand-alone command-line application that is used by an administrator to monitor the events from the IBM Content Manager library server.

The events are placed in a queue by the event monitor, which is monitored by the event handler to support FileNet Business Process Manager. If a custom handler is designed, applications from independent software vendors can also monitor the queue.

The event monitor connects to the IBM Content Manager database by using the IBM Content Manager administrator user ID and password. When the event monitor is started, it retrieves configuration data from the `cmbemconfig.properties` file. This configuration data consists of the initial context factory, the provider URL, the queue connection factory, and the queue name. The Java Naming and Directory Interface (JNDI) lookup then uses this information to connect to a Java Message Service (JMS) message queue, such as ICMMSGOQ.

Important: If you change the `cmbemconfig.properties` properties file, it does not affect event processing until after you restart the event monitor.

The event monitor retrieves event subscription information from the ICMSTEVENTSUBSCRIPTIONS and ICMSTEVENTSUBSCRIPTIONATTRS configuration tables. Event subscription information consists of the event type, item type, process name, process version, and process type.

Important: If you change the event subscription information, it does not affect event processing until after you restart the event monitor.

The event monitor scans the event table, with a frequency defined by the default scan interval, for any new events and processes each event sequentially. A JMS message is created to deliver the event information into the designated queue, such as, ICMMSGOQ. After the JMS message is sent successfully, the event in the table is marked as processed. All the events that are marked as processed are deleted from the event table according to the **`PURGE_INTERVAL`** parameter in `the cmbemconfig.properties` file.

Event data size

The EventDataC and EventDataB columns in the ICMSTEVENTQUEUE table are a pair of CLOB and BLOB data associated with the event. The amount of the CLOB and BLOB data that can be handled by the event monitor is limited to 512 KB for each column. If the data in either the EventDataC or the EventDataB columns exceed 512 KB, an error is logged in the log file. In addition, the event is marked for deletion and is not sent to the JMS queue. The event monitor then processes the next event.

Event monitor log file

You can set the location of the event monitor log file, `icmmonitor.log`, in the `cmblogconfig.properties` file. The `icmmonitor.log` file is located under the `log/em` subdirectory in the current working directory where an IBM Information Integrator for Content application is run. The following paths are example `icmmonitor.log` file locations for each operation system:

UNIX `/home/ibmcmadm/log/em`

Windows
`C:\IBM\db2cmv8\log\em`

"CBE format and JMS message queue"

Related information

Starting and stopping the event monitor

Modifying the event monitor and the event handler settings

Troubleshooting the event monitor and the event handler

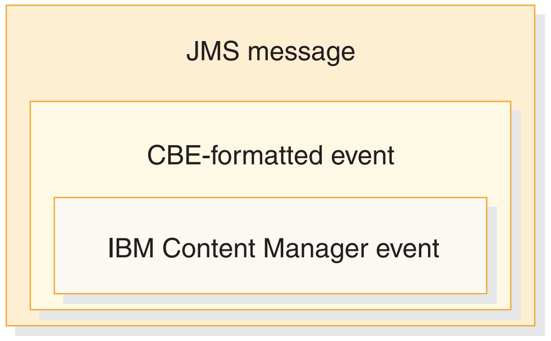
Working with the logging configuration file

CBE format and JMS message queue

To enable communication between the event monitor and the event handler, the event monitor converts IBM Content Manager events to Common Base Event (CBE) formatted events and sends those events to the event handler as Java Message Service (JMS) messages by using the JMS message queue. The event handler retrieves the JMS message from the message queue, parses the message, and retrieves the IBM Content Manager event from the JMS message.

An IBM Content Manager event contains both library server event information and event subscription information. A CBE-formatted event contains additional information about the host name, product name, and the environment. When an event occurs, such as the creation or modification of a document in an IBM Content Manager database, a library server event is generated. The event monitor retrieves the library server event from the event table, parses it, and combines it with event subscription information to form an IBM Content Manager event. The IBM Content Manager event is converted into a CBE-formatted event by adding the host name, product name, and environment information. The event monitor creates a JMS message from the CBE-formatted event and places it in the message queue. All events that are placed or received from the message queue must be in the form of a JMS message.

The following diagram shows the relationship between a JMS message, a CBE-formatted event, and an IBM Content Manager event:



"IBM Content Manager events"

"CBE XML element format" on page 293

"LDAP security authentication with JMS" on page 296

"Constructing IBM Content Manager events and creating JMS messages" on page 297

"Receiving a JMS error" on page 298

Related information

Setting up a JMS queue in MQ 6.0

Setting up a JMS queue with LDAP

JMS messages

IBM Content Manager events:

An IBM Content Manager event is generated by combining event subscription information with a library server event. The event monitor converts the IBM Content Manager event to a Common Base Event (CBE) formatted event, creates a Java Message Service (JMS) message from the CBE-formatted event and sends it to the JMS queue.

When an event occurs, such as the creation or modification of a document in the database, a library server event is generated. The event monitor retrieves the library server event from the event table, parses it, and combines it with event subscription information (for example, process names, process versions, process types, and mapping attributes) to form an IBM Content Manager event. Every IBM Content Manager event begins with the ICMEMSTART string and ends with the

ICMEMEND string. The event monitor then converts this event to a CBE-formatted event and sends it to the JMS queue as a JMS message.

"IBM Content Manager event formats"

"Examples of IBM Content Manager events for workflow requests" on page 292

IBM Content Manager event formats:

Each IBM Content Manager event has its own event format and event code.

IBM Content Manager supports the following event codes:

301	Event code for creating an item.
302	Event code for updating an item.
303	Event code for deleting an item.
306	Event code for reindexing an item.
307	Event code for adding an item to a folder (auto-enabled or manual).
308	Event code for removing an item from a folder (auto-enabled or manual).
309	Event code for checking in an item.
310	Event code for checking out an item.
311	Event code for reindexing an item when the reindex operation is monitored against the source item type.

IBM Content Manager events have the following formats:

Create item event format

The following event format is for an event that occurs when an item is created:

```
ICMEMSTART;ETYPE=ITEM;EACTION=CREATE;ECODE=301;
ITEMID=<item_id>;ITEMTYPE=<item_type_name>;
PID=<pid_string>;NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Update item event format

The following event format is for an event that occurs when an item is updated:

```
ICMEMSTART;ETYPE=ITEM;EACTION=UPDATE;ECODE=302;
ITEMID=<item_id>;ITEMTYPE=<item_type_name>;
OLDVERSION=<old_version>;
VERSION=<new_version>;
PID=<pid_string>;NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRVAL=<new_attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>; ATTRNAME=<attr_name>;
ATTRVAL=<new_attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Delete item event format

The following example shows the event format for an event that occurs when an item is deleted:

```
ICMEMSTART;ETYPE=ITEM;EACTION=DELETE;ECODE=303;
ITEMID=<item_id>;ITEMTYPE=<item_type_name>;
VERSION=<version>;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
PROCESS=<process_name>;PVERSION=<process_version>;...;
ICMEMEND
```

Reindex item event format

The following event format is for an event that occurs when an item is reindexed:

```
ICMEMSTART;ETYPE=FOLDER;EACTION=REINDEX;ECODE=306;
ITEMID=<item_id>;OLDITEMTYPE=<old_item_type_name>;
OLDVERSION=<version>;
ITEMTYPE=<new_item_type_name>;
VERSION=<new_version>;
PID=<pid_string>;NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;
ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;
ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
```

```
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Add item to folder event format

The following event format is for an event that occurs when an item is added to a folder:

```
ICMEMSTART;ETYPE=FOLDER;EACTION=ADD;ECODE=307;
ITEMID=<folder_item_id>;VERSION=<folder_version>;
CHILDITEMID=<child_item_id>;
CHILDVERSION=<child_version>;
PID=<pid_string>;
CHILDPID=<child_pid_string>;
NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Remove item from folder event format

The following event format is for an event that occurs when an item is removed from a folder:

```
ICMEMSTART;ETYPE=FOLDER;EACTION=REMOVE;ECODE=308;
ITEMID=<folder_item_id>;VERSION=<folder_version>;
CHILDITEMID=<child_item_id>;
CHILDVERSION=<child_version>;
PID=<pid_string>;
CHILDPID=<child_pid_string>;
NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
```

```
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Checkin item event format

The following event format is for an event that occurs when an item is checked in:

```
ICMEMSTART;ETYPE=ITEM;EACTION=CHECKIN;ECODE=309;
ITEMID=<item_id>;ITEMTYPE=<item_type_name>;
VERSION=<version>;
PID=<pid_string>;
CHECKINUID=<checkin_userid>;
CHECKINTIME=<checkin_time>;
CHECKOUTUID=<checkout_userid>;
CHECKOUTTIME=<checkout_time>;
OPERATIONTYPE=<operation_type>;
NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Checkout item event format

The following event format is for an event that occurs when an item is checked out:

```
ICMEMSTART;ETYPE=ITEM;EACTION=CHECKOUT;ECODE=310;
ITEMID=<item_id>;ITEMTYPE=<item_type_name>;
VERSION=<version>;
PID=<pid_string>;
CHECKOUTUID=<checkout_userid>;
CHECKOUTTIME=<checkout_time>;
OPERATIONTYPE=<operation_type>;
NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

| **Reindex from item event format**

| The following event format is for an event that occurs when an item is
| reindexed and the reindex operation is monitored against the source item
| type:

```
| ICMEMSTART;ETYPE=ITEM;EACTION=REINDEXFROM;ECODE=311;
| ITEMID=<item_id>;
| TOITEMTYPE=<to_item_type_name>;
| TOVERSION=<to_version>;
| ITEMTYPE=<item_type_name>;
| VERSION=<version>;
| PID=<pid_string>;
| NCOMPTYPE=<number_of_comp_type>;
| COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
| NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
| ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
| ATTRVAL=<attr_value>;...;
| COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
| NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
| ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
| ATTRVAL=<attr_value>;...;
| ETIME=<time_stamp>;
| NPROCESS=<number_of_process>;
| PROCESS=<process_name>;PVERSION=<process_version>;
| PTYPE=<process_type>;
| NMAPATTRNAME=<number_of_mapping_attr>;
| MAPATTRNAME=<mapping_attr_name>;
| CMATTRNAME=<cm_attr_name>;
| MAPATTRNAME=<mapping_attr_name>;
| CMATTRNAME=<cm_attr_name>;...;
| PROCESS=<process_name>;PVERSION=<process_version>;
| PTYPE=<process_type>;
| NMAPATTRNAME=<number_of_mapping_attr>;
```

```
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Additional information:

- The PID string is empty if the item is deleted before the event monitor has generated the PID string for the item.
- The `CMATTRNAME` attribute can have one of the following formats:
  - <cm_attr_name>: The attribute is a root attribute.
  - <comp_type_name>.<cm_attr_name>: The attribute is a subcomponent attribute.

The `OPERATIONTYPE` attribute for the checkin item and checkout item event formats specifies the type of operation that triggers the item to be checked in or checked out. It can have one of the following values:

1 Create an item.

2 Create a document part.

3 Update an item.

4 Update a document part.

5 Get an item.

6 Reindex an item (move one item type to another item type).

7 Get a work package.

8 Check in an item.

9 Check out an item.

10 Terminate a process.

11 Route to an end node. A checkin occurs because an end node is reached. When the end node is reached during a document routing process, an item that is checked out is checked in. One of the following scenarios can cause the end node to be reached:

- A collection of documents at a collection point node meets a requirement that results in an item route and the next node to be routed is an end node.
- An item route occurs at a decision point node and the next node to be routed is an end node.
- An item route is invoked and the next node to be routed is an end node

-1 Null (the operation type is not set).

The following five system-defined attributes are logged by the library server if they are subscribed to by the administrator. System-defined attributes of the root component are always used:

CREATEUSERID
Create user ID.

CREATETS
Create timestamp.

LASTCHANGEDUSERID
Update user ID.

LASTCHANGEDTS
Update timestamp.

EXPIRATIONDATE
Expiration timestamp.

Examples of IBM Content Manager events for workflow requests:

Each type of IBM Content Manager workflow request event has a different event format.

The following examples show the format of various types of IBM Content Manager workflow request events.

Important: If an IBM Content Manager attribute is an attribute of a child component, the component-type name of the child component is added as a prefix to the attribute name, for example, `CMATTRNAME=myChildComponent.SOURCE`.

Single workflow request event

```
ICMEMSTART;ETYPE=ITEM;EACTION=CREATE;ECODE=301;
ITEMID=A1001001A07C20A02957G14160;
ITEMTYPE=NOINDEX;
PID=86 3 ICM8 icmnlsdb7 NOINDEX59 26
  A1001001A07C20A02957G1416018
  A07C20A02957G141601 14 1000;
NCOMPTYPE=1;
COMPTYPE=NOINDEX;COMPNUMBER=1;
NATTRS=3;ATTRNAME=SOURCE;
ATTRLEN=32;ATTRTYPE=448;
ATTRVAL=Pooh;
ATTRNAME=USER_ID;
ATTRLEN=32;ATTRTYPE=448;
ATTRVAL=Peter;
ATTRNAME=TIMESTAMP;
ATTRLEN=26;ATTRTYPE=392;
ATTRVAL=2006-11-02 12:00:00.000000;
ETIME=2008-05-05 16:58:30.000000;
NPROCESS=1;
PROCESS=WF1;PVERSION=1.1;
PTYPE=1;
NMAPATTRNAME=2;
MAPATTRNAME=BPM1;
CMATTRNAME=SOURCE;
MAPATTRNAME=BPM2;
CMATTRNAME=USER_ID;
ICMEMEND
```

Multiple workflow requests event

```
ICMEMSTART;ETYPE=ITEM;EACTION=CREATE;ECODE=301;
ITEMID=A1001001A07C20A02957G14160;
ITEMTYPE=NOINDEX;
PID=86 3 ICM8 icmnlsdb7 NOINDEX59 26
  A1001001A07C20A02957G1416018
  A07C20A02957G141601 14 1000;
NCOMPTYPE=1;
COMPTYPE=NOINDEX;COMPNUMBER=1;
NATTRS=3;ATTRNAME=SOURCE;
ATTRLEN=32;ATTRTYPE=448;
ATTRVAL=Pooh;
ATTRNAME=USER_ID;
```

```
ATTRLEN=32;ATTRTYPE=448;
ATTRVAL=Peter;
ATTRNAME=TIMESTAMP;
ATTRLEN=26;ATTRTYPE=392;
ATTRVAL=2006-11-02 12:00:00.000000;
ETIME=2008-05-05 16:58:30.000000;
NPROCESS=2;
PROCESS=WF1;PVERSION=1.1;
PTYPE=1;
NMAPATTRNAME=2;
MAPATTRNAME=BPM1;
CMATTRNAME=SOURCE;
MAPATTRNAME=BPM2;
CMATTRNAME=USER_ID;
PROCESS=WF2;PVERSION=1.3;
PTYPE=1;
NMAPATTRNAME=1;
MAPATTRNAME=BPM1;
CMATTRNAME=TIMESTAMP;
ICMEMEND
```

Zero workflow requests event

```
ICMEMSTART;ETYPE=ITEM;EACTION=CREATE;ECODE=301;
ITEMID=A1001001A07C20A02957G14160;
ITEMTYPE=NOINDEX;
PID=86 3 ICM8 icmnlsdb7 NOINDEX59 26
  A1001001A07C20A02957G1416018
  A07C20A02957G141601 14 1000;
NCOMPTYPE=1;
COMPTYPE=NOINDEX;COMPNUMBER=1;
NATTRS=3;ATTRNAME=SOURCE;
ATTRLEN=32;ATTRTYPE=448;
ATTRVAL=Pooh;
ATTRNAME=USER_ID;
ATTRLEN=32;ATTRTYPE=448;
ATTRVAL=Peter;
ATTRNAME=TIMESTAMP;
ATTRLEN=26;ATTRTYPE=392;
ATTRVAL=2006-11-02 12:00:00.000000;
ETIME=2008-05-05 16:58:30.000000;
ICMEMEND
```

CBE XML element format:

The Common Base Event (CBE) specification defines an XML-based mechanism for managing events in an enterprise environment. Each CBE-formatted events is composed of XML elements.

The event monitor converts each IBM Content Manager event to a CBE-formatted event. Each CBE-formatted event contains the following XML element format:

CommonBaseEvent element
: Represents a CBE-formatted event. This element is the main root element of the CBE XML schema and has the following attributes:

  creationTime
  : Creation time of the event in GMT. For example, 2008-03-19T01:03:11.256Z.

  extensionName
  : The name of the event class that this event represents. The `extensionName` attribute is set to `CMEvent`.

globalInstanceId
: The UUID of a CBE-formatted event. The global instance ID of the CBE-formatted event is constructed as a combination of an item ID (26 alphanumeric characters), an event ID (19 digits), and an application type (three digits). For example, AA12345678901234567890123456789012345678901230O1. Both the item ID and the event ID are retrieved from the library server event. The supported application types are general integration ("000") for those events that are not associated with any application, and process integration ("001") for FileNet Business Process Manager.

priority
: The priority of the CBE-formatted event, which can range 1-100 (highest priority). The default value is `100`.

version
: The version of the CBE-formatted event as defined by the system. The `version` attribute is set to `1.0.1`.

contextDataElements

Represents the data referenced by the CBE-formatted event, which can be used to diagnose problems by correlating messages or generated events. This element has the following attributes:

name The name of the application that creates this context data element. The `name` attribute is set to `cmevent`.

type The data type of event data held by the `contextValue` attribute. For example, `string`.

contextValue
: The Content Manager event data. The length of the value of the `contextValue` element is limited to 1024 bytes. If the length of the Content Manager event is more than 1024 bytes, it is divided into multiple 1024-byte segments, each of which is represented by a separate `contextDataElements` element.

sourceComponentId

Represents the component that is affected by the event. This element has the following attributes:

application
: The name of the application that generates the event. The `application` attribute is set to `Content Manager Event Monitor`.

component
: The identity of the component that generates the event. For example, `Content Manager V8.4.01.000`.

componentIdType
: The ID that specifies the type of the component. The `componentIdType` attribute is set to `ProductName`.

executionEnvironment
: The operating environment, including the operating system and architecture. For example, `Windows XP[x86]`. The `executionEnvironment` value is provided by the Java Runtime Environment (JRE).

instanceId
The ID of the instance of the component that generates the event. The instance ID is always 1.

location
The host name and IP address of the event monitor. For example, `myhost/9.30.44.123`.

locationType
The format and meaning of the value of the `location` attribute. The `locationType` attribute is set to `Hostname`.

subComponent
The name of the subcomponent. The `subComponent` attribute is set to `Event Monitor`.

componentType
The common name of the component that generates the event. The `componentType` attribute is set to `Content Manager`.

situation
Represents the data that describes the situation reported by the event. This element has the following attribute:

categoryName
The category of the type of the situation that causes the event to be reported. The `categoryName` attribute is set to `OtherSituation`.

situationType

Represents the type or category of the situation that causes the event to be reported. The `situationType` attribute is set to `Application Event`.

This element has the following attributes:

xmlns:xsi
The XML schema namespace. The `xmlns:xsi` attribute is set to `http://www.w3.org/2001/XMLSchema-instance`.

xsi:type
The element type. The `xsi:type` attribute is set to `OtherSituation`.

reasoningScope
The scope of the situation reported, which defines whether this situation has an internal or external impact. The `reasoningScope` attribute is set to `EXTERNAL`.

"Example of converting an IBM Content Manager event to a CBE-formatted event"

Example of converting an IBM Content Manager event to a CBE-formatted event:

An IBM Content Manager event is converted to a Common Base Event (CBE) formatted event. A CBE-formatted event contains additional information about the host name, product name, and the environment.

The following example shows how an IBM Content Manager event is converted to a CBE-formatted event. Information in the CBE-formatted event is entered by the event monitor:

```
<CommonbaseEvent creationTime="2008-03-19T01:03:11:256Z"
extensionName="CMEvent"
globalInstanceId="AA123456789012345678901234567890123456789012300l"
priority="100" version="1.0.1">
```

```
 <contextDataElements name="cmevent" type="string">
  <contextValue>placeholder for Content Manager event</contextValue>
 </contextDataElements>
 <sourceComponentId application="Content Manager Event Monitor"
  component="Content Manager V8.4.01.000"
  componentIdType="Productname"
  executionEnvironment="Windows XP[x86]"
  instanceId="1"
  location="myhost/9.30.44.123"
  locationType="Hostname"
  subComponent="Event Monitor"
  componentType="Content Manager"/>
 <situation categoryName="OtherSituation">
  <situationType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:type="OtherSituation" reasoningScope="EXTERNAL">
   Application Event
  </situationType>
 </situation>
</CommonBaseEvent>
```

LDAP security authentication with JMS:

All users that use LDAP security authentication must be authenticated through a LDAP server before they are allowed to retrieve the JMS context objects for connecting to the JMS queue.

The event monitor uses the JMS queue to deliver the events to the event handler. By using the LDAP authentication mechanism, the JMS queue is protected from unauthorized access. No authentication is provided if a file system context, such as `com.sun.jndi.fscontext.RefFSContextFactory`, is used.

The event monitor and event handler use the LDAP authentication in the following sequence:

1. The event monitor and the event handler check the type of context factory in the `cmbemconfig.properties` file.
2. If the JMS is using LDAP context, such as `com.sun.jndi.ldap.LdapCtxFactory`, the authentication mechanism is enabled. The administrator is prompted for the LDAP user name and password.
3. The event monitor and the event handler call the JNDI API with LDAP principal and credentials to authenticate the user. If the authentication is successful, the event monitor or the event handler gets the JMS context objects, such as queue connection factory objects, and the queue object for establishing a valid connection to a JMS queue.

Properties for LDAP authentication

The `cmbemlogconfig.properties` file has two properties that are used for LDAP authentication.

LDAP_BASE_DN
Specifies the base DN (distinguished name) to be used for the LDAP user, for example, CN=Users, DC=svl, DC=ibm, DC=com.

LDAP_PREFIX
Specifies the prefix of a LDAP object, for example, cn=.

To construct a valid DN (distinguished name), the `LDAP_PREFIX` is added in front of the user name. The `LDAP_BASE_DN`, if specified, is appended at the end. For example, if the LDAP user name is *user1*, `LDAP_BASE_DN` is

`CN=Users,DC=svl,DC=ibm,DC=com` and the `LDAP_PREFIX` is `cn=`, the DN is constructed as `cn=user1,CN=Users,DC=svl,DC=ibm,DC=com`.

Constructing IBM Content Manager events and creating JMS messages:

The event monitor constructs the IBM Content Manager events, adds the Common Base Event (CBE) information as an XML string and creates a Java Message Service (JMS) message.

The event monitor constructs the IBM Content Manager events in the following sequence:

1. The event monitor retrieves the library server events that are generated from the event table.
2. The event monitor combines the event subscription information, such as process names, process version, process types, and mapping attributes, with the event data and converts the library server event to an IBM Content Manager event.
3. The event monitor adds the host name, product name, and environment information to the IBM Content Manager event and converts the IBM Content Manager event to a CBE-formatted event.
4. The event monitor creates a JMS message from the CBE-formatted event by setting the following JMS message properties:

   database name
   The name of the database that generates the event.

   type
   The type of the event. This property has one of following values: `item-create`, `item-update`, `item-delete`, `reindex`, `folder-add`, `folder-remove`, `checkin`, and `checkout`.

   launch indicator
   Indicates whether the message requests to start a workflow. The launch indicator is either `true` or `false`.

   message ID
   The same identifier as the global instance ID. For example, AA1234567890123456789012345678901234567890123001.

   application type
   The application associated with the workflow. If there is no application type, the application type is set to `none`. For FileNet Business Process Manager, the application type is `BPM`.

   data element count
   The number of context data elements in the CBE-formatted event that is extracted from the JMS message. If the length of the IBM Content Manager event is less than or equal to 1024 bytes, the data element count is set to 1 and the CBE event contains only one context data element. However if the count is greater than 1, multiple context data elements exist, which the event handler must concatenate.
5. The event monitor adds the JMS message to the JMS queue.
6. The event monitor marks each event as processed after a CBE-formatted event is created and sent to the JMS queue.
7. The JMS message is sent to the event handler through the JMS queue.

If the message is sent successfully, a database commit and a queue commit are performed. If the message cannot be composed because the database update failed, the event is processed in the next iteration. However, if the message cannot be delivered, a queue rollback is performed and another database update is performed to reset the flag to mark the message as not processed.

If a library server event contains multiple workflow requests for multiple application types (FileNet Business Process Manager or other applications), the event monitor generates one JMS message per application type. If an IBM Content Manager event has multiple application types, multiple JMS messages are generated that form a logical group. If one of the JMS messages in this group fails to be generated or delivered, a queue rollback is performed for all the messages that are associated with the library server event. The event handler responsible for the specific application type processes multiple workflow requests in the event.

Related information

Setting up a JMS queue in MQ 6.0

Setting up a JMS queue with LDAP

Receiving a JMS error:

You might receive a Java Message Service (JMS) error if JMS is not connected properly.

Symptom

JMS is not running.

Possible cause

JMS might not be constructed properly or might not be operational.

Action

Ensure that JMS is running.

Related information

Setting up a JMS queue in MQ 6.0

Setting up a JMS queue with LDAP

JMS messages

Event handler

The event handler gets the Java Message Service (JMS) message from the JMS queue, extracts the Common Base Event (CBE) formatted event, and calls the FileNet Business Process Manager API to start the workflow based on the event data that is contained in the CBE formatted event.

The administrator starts the event handler at the same time as the event monitor. The event handler connects to the FileNet Business Process Manager server. After the event handler is started, it retrieves the initial context factory, the provider URL, the queue connection factory, and the queue name from the `cmbemconfig.properties` file that are used for Java Naming and Directory Interface (JNDI) lookup for connecting to the JMS queue, such as ICMMSGOQ.

The event handler also retrieves the *RETRY_COUNT* value from the `cmbemconfig.properties` file. *RETRY_COUNT* denotes the number of attempts it will make to start the workflow.

When the event monitor adds the JMS message to the JMS queue, the event handler retrieves the JMS message from that queue and parses the JMS message to retrieve the workflow information from the message.

Event handler log file

The event handler log file is the `icmhandler.log` file. You can set the options for logging by modifying the `cmbemconfig.properties` file. You can also change the default path of the `icmhandler.log` file in the `cmbemconfig.properties` file.

"Parsing CBE-formatted events in JMS messages"

"FileNet Business Process Manager workflow and the VWAttachment object" on page 300

"Attribute mappings for the event handler" on page 300

"Event handler throughput and event serialization" on page 301

Related information

Starting and stopping the event handler

Troubleshooting the event monitor and the event handler

Working with the logging configuration file

Parsing CBE-formatted events in JMS messages

The event handler retrieves the Java Message Service (JMS) message from the JMS queue and parses the JMS message to retrieve the Common Base Event (CBE) formatted event.

After the JMS message is generated by the event monitor, the message is sent to the JMS queue. The event handler retrieves the message from the JMS queue and parses the message to get the event data. The event data in the message specifies the FileNet Business Process Manager workflow process to be started.

The event handler parses the JMS message in the following sequence:

1. The event handler retrieves the Common Base Event (CBE) formatted event from the JMS message. For example:
   ```
   <contextDataElements name="cmevent" type="string">
    <contextValue>placeholder for Content Manager event</contextValue>
   </contextDataElements>
   ```
   Important: If the length of the IBM Content Manager event is less than or equal to 1024 bytes, the CBE-formatted event contains only one context data element with the multiple context data elements concatenated.
2. The event handler parses the string within the contextValue XML element of the CBE-formatted event based on the event format in the IBM Content Manager events.
3. The workflow information, such as the process name, process version, and the mapping attributes are retrieved from the event data.
4. The event handler starts a workflow after mapping the IBM Content Manager attributes of the item types to the process attributes of the FileNet Business Process Manager. For single-valued attributes, the data type of integer, string, double, date, and time stamp (java.util.Date) are mapped because the data

types are supported by both IBM Content Manager and FileNet Business Process Manager. For multi-valued attributes that are applicable to child components in IBM Content Manager, the attribute values in the child components are considered as values of a multi-valued attribute. The values of a multi-valued attribute are mapped to a composite data type (array) of simple data types even though there is only one value of the attribute.

For each incoming JMS message, the event handler starts a workflow. If the FileNet Business Process Manager fails to start a workflow, the event handler attempts to start the workflow again based on the retry number. The default value of the retry number is three. However, you can configure the retry number depending on your requirements.

Important: The FileNet Business Process Manager workflow definition must be transferred to the Process Engine before it can be used by the event handler. The workflow information in the event subscription must be consistent with the workflow definition in the FileNet Business Process Manager server. If the workflow definition is changed, the change to the initial flag of the attachment is not effective until the event handler is restarted.

Related reference

IBM Content Manager events

FileNet Business Process Manager workflow and the VWAttachment object

If the attachment name can be retrieved from the workflow definition, the event handler creates a `VWAttachment` object with the information that is used to start a FileNet Business Process Manager workflow.

The event monitor creates a valid PID string of the item being monitored in the event and passes it to the event handler in the Java Message Service (JMS) message. The PID string is saved in the GUID field of the `VWAttachment` object. If the item is deleted after the event has been logged, the event monitor does not generate a PID string and the PID string is set to empty (“”) in the GUID field of the `VWAttachment` object.

The VWAttachment object that is generated by the event handler is intercepted by the Process Engine and, depending on the workflow definition, might be passed to the CE_Operations component.

Important: If the attachment name cannot be retrieved from the workflow definition, the `VWAttachment` object is not created when the workflow is started.

Attribute mappings for the event handler

To start a FileNet Business Process Manager process, the event handler maps the IBM Content Manager attribute types to the attribute data types that are supported by FileNet Business Process Manager.

Attention: Because the decimal type data in IBM Content Manager is converted to the float type data in FileNet Business Process Manager, there might be some loss of data.

The composite data type, such as an array of simple data types, is supported in FileNet Business Process Manager. The attribute values of the child components in IBM Content Manager are mapped to a composite data type in FileNet Business Process Manager. If no IBM Content Manager attribute is found during the mapping to a FileNet Business Process Manager process attribute, a workflow is started with partial process attributes. The IBM Content Manager attributes that are not found are not mapped to FileNet Business Process Manager process attributes.

Important: The user-specified date and time stamp will use the local time zone as the default value.

Table 41. Attribute mappings in the event handler

IBM Content Manager attribute types	FileNet Business Process Manager attribute types	Event handler format
Character	string	Java String
Variable character	string	Java String
Short integer	integer	Java Integer
Long integer	integer	Java Integer
Decimal	float	Java Double
Double	float	Java Double
Date	time	java.util.Date
Time	Not supported	Not applicable
Time stamp	time	java.util.Date
BLOB and CLOB	Not supported	Not applicable

Event handler throughput and event serialization

The event monitor generates the IBM Content Manager events in the same sequence in which the library server events are generated. Because events are processed serially, the throughput of the event handler can be affected.

Event serialization ensures event integrity. Because the event handler, which is a single-threaded FileNet Business Process Manager application, retrieves and processes the IBM Content Manager events serially for event integrity, the multi-threading feature of FileNet Business Process Manager is not used.

If your applications do not require that events be processes serially, you can use the multi-threading feature of FileNet Business Process Manager and start multiple instances of event handlers for higher throughput. For additional information, FileNet Business Process Manager documentation describes the best practices for managing process sequencing.

Managing event subscriptions

To manage event subscriptions, such as adding, deleting, or updating an event subscription, IBM Content Manager provides three classes: the `DKEventSubscriptionMgmtICM` class, the `DKEventSubscriptionDefICM` class, and the`DKEventSubscriptionAttrDefICM` class.

DKEventSubscriptionMgmtICM class

The `DKEventSubscriptionMgmtICM` class represents the event subscription management for the specified IBM Content Manager datastore and provides methods to add, delete, update, and list event subscriptions.

The following code example shows how to access the `DKEventSubscriptionMgmtICM` class:

```
DKDatastoreICM dsICM = new DKDatastoreICM();
DKDatastoreDefICM dsDef = null;
DKDatastoreAdminICM dsAdmin = null;
DKEventSubscriptionMgmtICM eventSubMgmt = null;

// connect to content manager
dsICM.connect("ICMNLSDB","icmadmin","password","");
dsDef = (DKDatastoreDefICM)dsICM.datastoreDef();
dsAdmin = (DKDatastoreAdminICM)dsDef.datastoreAdmin();
// get the event subscription management
eventSubMgmt = dsAdmin.eventSubscriptionManagement();
```

The `DKEventSubscriptionMgmtICM` class has the following methods:

add Adds event subscriptions for an item type.

```
void add(DKEventSubscriptionDefICM eventSub);
```

del Deletes event subscriptions for an item type.

```
void del(DKEventSubscriptionDefICM eventSub);
```

update
Updates event subscriptions for an item type.

```
void update(DKEventSubscriptionDefICM eventSub);
```

listEventSubscriptions
Lists the event subscriptions for an item type.

```
dkCollection listEventSubscriptions(String itemTypeName);
```

listEventSubscriptions
Lists all the event subscriptions in the library server.

```
dkCollection listEventSubscriptions();
```

DKEventSubscriptionDefICM class

The `DKEventSubscriptionDefICM` class represents an event subscription definition in IBM Content Manager and provides methods to access and update event subscription information.

```
DKEventSubscriptionDefICM(dkDatastore ds);
```

The `DKEventSubscriptionDefICM` class contains the following methods:

setId Sets the event subscription ID. The event subscription ID is generated by the library server.

```
void setId(int id);
```

getId Gets the event subscription ID. The event subscription ID is generated by the library server.

```
int getId();
```

setEventCode
Sets the event code.

The valid event codes are:

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.UNDEFINED
Undefined event.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.ADD_ITEM
Add item.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.DELETE_ITEM
Delete item.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.UPDATE_ITEM
Update item.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.MOVE_ITEM_TO_ITEMTYPE
Move item to item type.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.ADD_ITEM_TO_FOLDER
Add item to folder.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.REMOVE_ITEM_FROM_FOLDER
Remove item from folder.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.CHECKIN
Check in an item.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.CHECKOUT
Check out an item.

```
void setEventCode
(DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE eventCode);
```

getEventCode
Gets the event code.

```
DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE getEventCode();
```

setItemTypeName
Sets the item type name.

```
void setItemTypeName(String itemTypeName);
```

getItemTypeName
Gets the item type name.

```
String getItemTypeName();
```

setProcessId
Sets the process ID.

```
void setProcessId(String processId);
```

getProcessId
Gets the process ID.

```
String getProcessId();
```

setProcessVersion
Sets the process version.

```
void setProcessVersion(String setProcessVersion);
```

getProcessVersion
Gets the process version.

```
String getProcessVersion();
```

setIntegrationType
Sets the event subscription integration type.

The valid integration types are:

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_ INTEGRATION_TYPE.UNDEFINED
Undefined integration type.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_ INTEGRATION_TYPE.PROCESS
Process integration type. The event subscription contains a process ID and event subscription attributes with attribute mappings. This type of integration is for Process Engine products such as FileNet Business Process Manager.

DKEventSubscriptionDefICM.DK_ICM_EVENT_SUBSCRIPTION_ INTEGRATION_TYPE.GENERAL
General integration type. The event subscription does not contain a process ID and contains event subscription attributes that do not have attribute mappings.

```
void setIntegrationType
(DK_ICM_EVENT_SUBSCRIPTION_INTEGRATION_TYPE integrationType);
```

getIntegrationType
Gets the event subscription integration type.

```
DK_ICM_EVENT_SUBSCRIPTION_INTEGRATION_TYPE getIntegrationType();
```

getCreatedTimestamp
Gets the time stamp when the event subscription was created in the library server.

```
DKTimestamp getCreatedTimestamp();
```

getUpdatedTimestamp
Gets the time stamp when the event subscription was updated in the library server.

```
DKTimestamp getUpdatedTimestamp();
```

getCreateUser
Gets the user who created the event subscription.

```
string getCreateUser();
```

setEventSubscriptionAttributes
Sets the event subscription attributes.

```
void setEventSubscriptionAttributes
   (dkCollection attrCollection);
```

getEventSubscriptionAttributes
Gets the event subscription attributes.

```
dkCollection getEventSubscriptionAttributes();
```

DKEventSubscriptionAttrDefICM class

The `DKEventSubscriptionAttrDefICM` class represents an event subscription attribute definition in IBM Content Manager and provides methods to access and update event subscription attribute information.

```
DKEventSubscriptionAttrDefICM(dkDatastore ds)
```

The `DKEventSubscriptionAttrDefICM` class contains the following methods:

setComponentTypeName
Sets the component type name.

```
void setComponentTypeName(String componentTypeName);
```

getComponentTypeName
Gets the component type name.

```
string getComponentTypeName();
```

setAttributeName
Sets the attribute name.

```
void setAttributeName(String attrName);
```

getAttributeName
Gets the attribute name.

```
string getAttributeName();
```

setAttributeGroupName
Sets the attribute group name.

```
void setAttributeGroupName(string attrGroupName);
```

getAttributeGroupName
Gets the attribute group name.

```
string getAttributeGroupName();
```

setMappingAttributeName
Sets the mapping attribute name.

```
void setMappingAttributeName(string mappingAttrName);
```

getMappingAttributeName()
Gets the mapping attribute name.

```
string getMappingAttributeName();
```

"Adding event subscriptions for item types"
"Deleting event subscriptions for item types" on page 306
"Updating event subscriptions for item types" on page 306
"Listing event subscriptions for item types" on page 307
"Event subscription methods" on page 307

Adding event subscriptions for item types

The DKEventSubscriptionMgmtICM class provides the add method to add the event subscriptions for an item type.

Example

The following example adds an event subscription for an item type:

```
DKDatastoreICM dsICM = new DKDatastoreICM();
DKDatastoreDefICM dsDef = null;
DKDatastoreAdminICM dsAdmin = null;
DKEventSubscriptionMgmtICM eventSubMgmt = null;
DKEventSubscriptionDefICM eventSub = null;
// connect to content manager
dsICM.connect("ICMNLSDB","icmadmin","password","");
dsDef = (DKDatastoreDefICM)dsICM.datastoreDef();
dsAdmin = (DKDatastoreAdminICM)dsDef.datastoreAdmin();
// get the event subscription management
eventSubMgmt = dsAdmin.eventSubscriptionManagement();
docItemType  = (DKItemTypeDefICM)dsDef.retrieveEntity("NOINDEX");
if (docItemType != null)
{// item type found
attrDef = docItemType.retrieveAttr("USER_ID");
if (attrDef != null)
 {
 // attribute found
 // Create a new event subscription
 eventSub = new DKEventSubscriptionDefICM(dsICM);
 eventSub.
  setEventCode(DKEventSubscriptionDefICM.
  DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.UPDATE_ITEM);
  eventSub.setItemTypeName("NOINDEX");
  eventSub.setIntegrationType(DKEventSubscriptionDefICM.
  DK_ICM_EVENT_SUBSCRIPTION_INTEGRATION_TYPE.PROCESS);
  eventSub.setProcessId("ProcessA");
  attrCol = new DKSequentialCollection();
  eventSubAttr = new DKEventSubscriptionAttrDefICM(dsICM);
  eventSubAttr.setAttributeName("USER_ID");
  eventSubAttr.setMappingAttributeName("Name");
  attrCol.addElement(eventSubAttr);
  // add event subscription
  eventSubMgmt.add(eventSub);
  }
}
dsICM.disconnect();
```

Deleting event subscriptions for item types

The `DKEventSubscriptionMgmtICM` class provides the `del` method to delete the event subscriptions for an item type.

Example

The following example deletes an event subscription for an item type:

```
DKDatastoreICM dsICM = new DKDatastoreICM();
DKDatastoreDefICM dsDef = null;
DKDatastoreAdminICM dsAdmin = null;
DKEventSubscriptionMgmtICM eventSubMgmt = null;
DKEventSubscriptionDefICM eventSub = null;
DKSequentialCollection pCol = null;
dkIterator pIter = null;
// connect to content manager
dsICM.connect("ICMNLSDB","icmadmin","password","");
dsDef = (DKDatastoreDefICM)dsICM.datastoreDef();
dsAdmin = (DKDatastoreAdminICM)dsDef.datastoreAdmin();
// get the event subscription management
eventSubMgmt = dsAdmin.eventSubscriptionManagement();
// list event subscriptions for an item type name
pCol = (DKSequentialCollection)eventSubMgmt.listEventSubscriptions("NOINDEX");
// iterate over the event subscriptions
pIter = pCol.createIterator();
while (pIter.more())
{
 eventSub = (DKEventSubscriptionDefICM)pIter.next();
  // delete event subscription
  eventSubMgmt.del(eventSub);
}
dsICM.disconnect();
```

Updating event subscriptions for item types

The `DKEventSubscriptionMgmtICM` class provides the `update` method to update the event subscriptions for an item type.

Example

The following example updates an event subscription for an item type:

```
DKDatastoreICM dsICM = new DKDatastoreICM();
DKDatastoreDefICM dsDef = null;
DKDatastoreAdminICM dsAdmin = null;
DKEventSubscriptionMgmtICM eventSubMgmt = null;
DKEventSubscriptionDefICM eventSub = null;
DKSequentialCollection pCol = null;
dkIterator pIter = null;
// connect to content manager
dsICM.connect("ICMNLSDB","icmadmin","password","");
dsDef = (DKDatastoreDefICM)dsICM.datastoreDef();
dsAdmin = (DKDatastoreAdminICM)dsDef.datastoreAdmin();
// get the event subscription management
eventSubMgmt = dsAdmin.eventSubscriptionManagement();
// list event subscriptions for an item type name
pCol = (DKSequentialCollection)eventSubMgmt.listEventSubscriptions("NOINDEX");
// iterate over the event subscriptions
pIter = pCol.createIterator();
while (pIter.more())
{
 eventSub = (DKEventSubscriptionDefICM)pIter.next();
 // change event code of the event subscription added in earlier example
 // to add item event code.
 eventSub.setEventCode(DKEventSubscriptionDefICM.
DK_ICM_EVENT_SUBSCRIPTION_EVENT_CODE.ADD_ITEM);
 // update event subscription
 eventSubMgmt.update(eventSub);
}
dsICM.disconnect();
```

Listing event subscriptions for item types

The `DKEventSubscriptionMgmtICM` class provides the `listEventSubscriptions` method to list the event subscriptions for an item type.

Example

The following example lists the event subscriptions for an item type:

```
DKDatastoreICM dsICM = new DKDatastoreICM();
DKDatastoreDefICM dsDef = null;
DKDatastoreAdminICM dsAdmin = null;
DKEventSubscriptionMgmtICM eventSubMgmt = null;
DKEventSubscriptionDefICM eventSub = null;
DKSequentialCollection pCol = null;
dkIterator pIter = null;

// connect to content manager
dsICM.connect("ICMNLSDB","icmadmin","password","");
dsDef = (DKDatastoreDefICM)dsICM.datastoreDef();
dsAdmin = (DKDatastoreAdminICM)dsDef.datastoreAdmin();
// get the event subscription management
eventSubMgmt = dsAdmin.eventSubscriptionManagement();
// list event subscriptions for an item type name
pCol =
(DKSequentialCollection)eventSubMgmt.listEventSubscriptions("NOINDEX");
// iterate over the event subscriptions
pIter = pCol.createIterator();
while (pIter.more())
 {
 eventSub = (DKEventSubscriptionDefICM)pIter.next();
 }

 dsICM.disconnect();
```

Event subscription methods

IBM Content Manager provides four event subscription methods for library server configuration and two event subscription methods for item type definition. You can use the event subscription methods to enable event subscriptions and check whether the event subscription is enabled for the library server an item type.

Event subscription methods for the library server

For the library server to generate events for a particular event subscription, the event subscriptions must be enabled for the library server and the specific item type.

The `DKLSCfgDefICM` class contains the following event subscription methods for the library server:

isEventSubscriptionsEnabled
Checks whether the event subscriptions indicator is enabled.

```
boolean isEventSubscriptionsEnabled();
```

setEventSubscriptionsEnabled
Sets the event subscriptions enabled flag to on or off.

```
void setEventSubscriptionsEnabled(boolean bTurnOn);
```

isEventMonitorActive
Checks whether the event monitor is active.

```
boolean isEventMonitorActive();
```

setEventMonitorActive
Sets the event monitor active flag to on or off.

```
void setEventMonitorActive(boolean bTurnOn);
```

Event subscription methods for item type definitions

The `DKItemTypeDefICM` class contains the following event subscription methods for item type definition:

isEventSubscriptionsEnabled
Checks whether the event subscriptions indicator is enabled.

```
boolean isEventSubscriptionsEnabled();
```

setEventSubscriptionsEnabled
Sets the event subscriptions enabled flag to on or off.

```
void setEventSubscriptionsEnabled(boolean bTurnOn);
```

Configuring IBM Content Manager to use the FileNet Business Process Manager workflow

To develop an application that uses FileNet Business Process Manager workflow with IBM Content Manager, you must first configure IBM Content Manager by adding the JAR files to the class path of the Component Manager of the FileNet Business Process Manager. You must also add IBM Content Manager login modules to the login module configuration file of FileNet Business Process Manager.

1. “Updating the class path of the Component Manager of FileNet Business Process Manager” on page 309
2. “Adding the IBM Content Manager login modules” on page 310

Related information

Software requirements

Updating the class path of the Component Manager of FileNet Business Process Manager

Before you can use a Component Manager instance for the CE_Operations component, you must update the class path for the Component Manager instance of FileNet Business Process Manager with the directory and the JAR files for IBM Content Manager. The Component Manager is installed on the Application Engine server.

Ensure that the following products are installed:

- IBM Content Manager Version 8.4.2
- FileNet Business Process Manager Version 4.5.1
- IBM Information Integrator for Content Version 8.4.2 on FileNet Business Process Manager Application Engine server

For more information about the Application Engine, in the FileNet P8 Documentation table of contents, click **FileNet P8 Administration** > **Application Engine Administration**.

To add the directory and `JAR` files to the class path of the Component Manager:

1. Open the `taskman.properties` file in *INSTALL_HOME*/FileNet/AE/Router, where *INSTALL_HOME* is the directory path where Application Engine is installed.
2. In the taskman.properties file, find the `TaskManager.ComponentManager.ClassPath` line and add the following directory and `JAR` files to the beginning of the class path:

   UNIX *`IBMCMROOT`*`/cmgmt:` *`IBMCMROOT`*`/lib/cmbcm81.jar:` *`IBMCMROOT`*`/lib/cmbicm81.jar:` *`IBMCMROOT`*`/lib/jcache.jar:` *`IBMCMROOT`*`/lib/cmblog4j81.jar:` *`IBMCMROOT`*`/lib/log4j-1.2.15.jar:` *`IBMCMROOT`*`/lib/db2jcc_license_cisuz.jar:` *`IBMCMROOT`*`/lib/db2jcc.jar:` *`IBMCMROOT`*`/lib/db2jcc_license_cu.jar:`

   IBMCMROOT is the location where IBM Content Manager and IBM Information Integrator for Content are installed.

   For Oracle, add the *`ORACLE_HOME`*`/jdbc/lib/ojdbc14.jar` file to the class path

   Windows
   `IBMCMROOT``\cmgmt;` *`IBMCMROOT`*`\lib\cmbcm81.jar;` *`IBMCMROOT`*`\lib\cmbicm81.jar;` *`IBMCMROOT`*`\lib\jcache.jar;` *`IBMCMROOT`*`\lib\cmblog4j81.jar;` *`IBMCMROOT`*`\lib\log4j-1.2.15.jar;` *`IBMCMROOT`*`\lib\db2jcc_license_cisuz.jar;` *`IBMCMROOT`*`\lib\db2jcc.jar;` *`IBMCMROOT`*`\lib\db2jcc_license_cu.jar;`

   IBMCMROOT is the location where IBM Content Manager and IBM Information Integrator for Content are installed.

   For Oracle, add the *`ORACLE_HOME`*`\jdbc\lib\ojdbc14.jar` file to the class path

   Important: Only new Component Manager instances inherit the new class path. If you already have an existing Component Manager instance and do not want to create a new one, you can use Process Task Manager to add the new class path for the directory and JAR files of IBM Content Manager to the class path of the existing Component Manager instance.

For more information about Component Manager, in the FileNet P8 Documentation table of contents, click **FileNet P8 Administration** > **Enterprise-wide Administration** > **Process Task Manager** > **Component Manager**.

Related information

Example sample configuration

Adding the IBM Content Manager login modules

For authentication, both IBM Content Manager and FileNet Business Process Manager must use the same Java Authentication and Authorization Server (JAAS) callback handler. To configure IBM Content Manager and FileNet Business Process Manager to use the same JAAS credentials, add the IBM Content Manager login module to the `taskman.login.config` file.

Ensure that the following products are installed:
- IBM Content Manager Version 8.4.2
- FileNet Business Process Manager Version 4.5.1
- IBM Information Integrator for Content Version 8.4.2 on FileNet Business Process Manager Application Engine server

To use the `DKContentOperationsICM` class, you must install both IBM Content Manager and FileNet Business Process Manager on the same machine.

Login modules are identified and located by the CELogin section of the `taskman.login.config` file, which is created as part of the FileNet Business Process Manager installation. CELogin identifies the set of login modules used by the CE_Operations component, which is created as a part of a FileNet Business Process Manager installation. When a new Component Manager instance is created, the CE_Operations component is displayed as a component queue node in the Process Configuration Console window or in the Process Task Manager windows.

For more information about Application Engine, in the FileNet P8 Documentation table of contents, click **FileNet P8 Administration** > **Application Engine Administration**.

For documentation about how to configure Component Queues, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Configuration Console** > **Queues** > **Configure Component Queues**.

To configure FileNet Business Process Manager login modules for IBM Content Manager:

1. Open the `taskman.login.config` file in *INSTALL_HOME*/FileNet/AE/Router, where *INSTALL_HOME* is the directory path in which Application Engine is installed.
2. In the `taskman.login.config` file, add the `DKLoginModuleICM` information to the `CELogin` section:
   a. Specify the `serversList` option. The `serversList` option specifies the list of library servers for an IBM Content Manager configuration.

b. Optional: Specify the `queryMaxResultsList` option. The `queryMaxResultsList` option indicates the maximum number of results that are returned from the `searchForMany` method of the `DKContentOperationsICM` class.

   If you specify the `queryMaxResultsList` option, it must match the number of servers that are specified for the library server list in the `serversList` option; otherwise, an error is generated. If the `queryMaxResultsList` option is not specified, a default value of 50 is used by the `searchForMany` method.

   Important: If the `queryMaxResultsList` option is set to 0, all of the results are returned.

Examples of DKLoginModuleICM in CELogin

The following examples show how you can add the DKLoginModuleICM information in the CELogin section in the `taskman.login.config` file:

Single library server with the `serversList` option specified

```
filenet.vw.server.VWLoginModule required;
filenet.contentops.ceoperations.util.
CELoginModule required credTag=Clear;
com.ibm.mm.sdk.jaas.DKLoginModuleICM required
  serversList="icmnlsdb";
```

The library server name is *"icmnlsdb"*.

Multiple library servers with the `serversList` option specified

```
filenet.vw.server.VWLoginModule required;
filenet.contentops.ceoperations.util.
CELoginModule required credTag=Clear;
com.ibm.mm.sdk.jaas.DKLoginModuleICM required
  serversList="icmnlsdb, icmnlsdb2";
```

The library server names are *"icmnlsdb"* and *"icmnlsdb2"*.

Single library server with the `queryMaxResultsList` option specified

```
filenet.vw.server.VWLoginModule required;
filenet.contentops.ceoperations.util.
CELoginModule required credTag=Clear;
com.ibm.mm.sdk.jaas.DKLoginModuleICM required
  serversList="icmnlsdb"
queryMaxResultsList="25";
```

The library server name is *"icmnlsdb"*.

Multiple library server with the `queryMaxResultsList` option specified

```
filenet.vw.server.VWLoginModule required;
filenet.contentops.ceoperations.util.
CELoginModule required credTag=Clear;
com.ibm.mm.sdk.jaas.DKLoginModuleICM required
  serversList="icmnlsdb,icmnlsdb2"
queryMaxResultsList="25,50";
```

The library server names are *"icmnlsdb"* and *"icmnlsdb2"*.

Related information

Example sample configuration

Designing FileNet Business Process Manager workflow applications

To develop a FileNet Business Process Manager process application, you must first configure queues and regions and use the Process Configuration Console to check whether the same user name and password is used for both IBM Content Manager and FileNet Business Process Manager. Then, use the Process Designer to design your business processes and use the IBM Content Manager and FileNet Business Process Manager APIs to develop a workflow application.

The Process Configuration Console provides the tools to manage the workflow database configuration, such as isolated regions and queues. Before you start designing your workflow application, use the Process Configuration Console to initialize and connect regions and configure different types of queues.

For documentation for developers, in the FileNet P8 Documentation table of contents, click **Developer Help** > **Process Engine Development** > **Developer's Guide** > **Developing Process Applications**.

For documentation about how to use the Process Configuration Console, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Configuration Console**.

1. "Developing custom applications"
2. "Configuring and managing queues by using the Process Configuration Console"
3. "Managing isolated regions by using the Process Configuration Console" on page 313
4. "Checking whether the user name and the password are the same for both IBM Content Manager and FileNet Business Process Manager" on page 314
5. "Developing a workflow definition by using the Process Designer" on page 314

Developing custom applications

You can use the sample code that is packaged with IBM Content Manager to develop your own custom applications.

A sample workflow application that was developed by using the FileNet Business Process Manager and IBM Content Manager is provided. The source files for the sample are located in `%IBMCMROOT%/samples/bpm/cm8bpm/java` directory. See the Samples Readme file in this directory for details about the sample and the setup requirements.

Configuring and managing queues by using the Process Configuration Console

The queues hold the work items that are waiting to be processed. There are three types of queues: user queues, work queues, and component queues. You can manage the user queues and configure the work and component queues by using the Process Configuration Console.

Each user has an Inbox that holds work items that are assigned to that user. A user might also have a queue of Tracker items. The Inbox and Tracker queues are created automatically during initialization of the isolated region.

A work queue holds work items that can be completed by one or a number of users rather than by a specific participant, or work items that can be completed by an automated process. In a workflow definition, the workflow author can assign steps to a specific work queue.

The component queues process a workflow step by using either a Java adaptor or a Java Message Service (JMS) adaptor.

For documentation about how to manage queues, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Configuration Console** > **Queues**.

To manage and configure queues:

1. Modify the user queue properties, such as description, system and data fields, indexes, and users' privileges for accessing the queue in the Process Configuration Console.
2. Create and configure work queues by specifying the queue properties in the Process Configuration Console.
3. Configure the component queues by using the Process Configuration Console to process a workflow step.

   For documentation on how to configure work queues, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Configuration Console** > **Queues** > **Configure component queues**.

Managing isolated regions by using the Process Configuration Console

An isolated region is a logical subdivision of the workflow database that contains the queues for the work items and other configuration information.

In the Process Configuration Console, you connect to the isolated region and initialize it with default structures, such as an Inbox, to process the workflow. You can also customize the isolated region by creating work queues to hold work items that are waiting to be processed.

For documentation on how to manage regions, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Configuration Console** > **Isolated regions**.

To manage isolated regions:

1. Initialize an isolated region:
   a. In the Process Configuration Console window, select the isolated region that you want to initialize.
   b. Click **Action** > **Initialize Isolated Region**.

   For documentation on how to initialize isolated regions, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Configuration Console** > **Isolated regions** > **Initialize an isolated region**.
2. Connect to a region:
   a. In the Process Configuration Console window, select the isolated region that you want to connect.
   b. Click **Connect** on the toolbar.

For documentation on how to connect to regions, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Configuration Console** > **Isolated regions** > **Connect to an isolated region**.

Checking whether the user name and the password are the same for both IBM Content Manager and FileNet Business Process Manager

The user name and password must be the same for both FileNet Business Process Manager and IBM Content Manager because they use the same Java Authentication and Authorization Server (JAAS) callback handler for authentication.

To check the user name and the password that are used in FileNet Business Process Manager, follow these steps:

1. Open a browser to the following Web address: `http://<Application Engine name:portnumber>/Workplace`.
2. In the Workplace client, click the **Admin** tab.
3. On the Admin page, click **Process Configuration Console**.
4. From the **Component Queues** folder icon, click the **CE_Operations** component queue, then double-click the **CE_Operations** component queue.
5. In the Component Properties window, click the **Adapter** tab to open the Adapter page.
6. In the **JAAS Credentials** section, ensure that the user name and password are same as the user name and password in IBM Content Manager.

   Important: IBM Content Manager must have a user name and password that matches a user name and password in FileNet Business Process Manager. If it does not, you can create a matching user name and password in IBM Content Manager by using the system administration client.

Developing a workflow definition by using the Process Designer

Use the Process Designer to develop a workflow definition. A workflow definition is a processing template that specifies the steps, resources, and the routing logic that is needed to complete a business process.

To enable users to access FileNet Business Process Manager workflow during run time, the workflow administrators or the application developers assign the users' work from the Process Designer.

To develop a workflow application, define your workflow according to your business requirements and configure the `DKContentOperationsICM` methods. Then, create your workflow definition by using the Process Designer. You can use the Process Designer to open a workflow application that you created and saved previously or search for documents in the workflow. You can also develop custom applications by using the samples that are provided by IBM Content Manager.

For documentation about how to use the Process Designer, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer**.

1. "Defining a FileNet Business Process Manager workflow" on page 315

2. “Creating a FileNet Business Process Manager workflow definition by using the CE_Operations component” on page 349
3. “Tracking the progress of a workflow application” on page 351
4. “Opening a workflow by using the Process Designer” on page 352
5. “Searching for documents in FileNet Business Process Manager workflow” on page 352

Related reference

DKContentOperationsICM class

Defining a FileNet Business Process Manager workflow

A workflow definition is a representation of the activities that are required to accomplish a business process. You create your workflow after analyzing your business processes.

The workflow definition acts as a processing template for routing the work, data, attachments, and other information that is required to complete the activities to the specified participants and automated processes.

For documentation about how to use the Process Designer to define a workflow, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Getting Started** > **About defining a workflow**.

To define a workflow:

1. Analyze your business requirements to break the process down into units that correspond with workflow definition components.
2. Create the workflow by using the Process Designer.
3. Link the workflow definitions.

DKContentOperationsICM class

To enable FileNet Business Process Manager to access documents in the IBM Content Manager database, IBM Content Manager provides the `DKContentOperationsICM` class, which implements the `IContentOperations` Java interface for the component integrator in FileNet Business Process Manager. The `DKContentOperationsICM` class uses Java Authentication and Authorization Service (JAAS) for authentication.

You can use the `DKContentOperationsICM` methods through the Process Designer of FileNet Business Process Manager. You must install both IBM Content Manager and FileNet Business Process Manager on the same machine to use the `DKContentOperationsICM` class.

For documentation about how to use the Process Designer, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer**.

“VWAttachment object”

“Attribute mappings for the DKContentOperationsICM class” on page 316

“Supported DKContentOperationsICM class methods in IBM Content Manager” on page 318

“Unsupported DKContentOperationsICM class methods in IBM Content Manager” on page 344

VWAttachment object:

IBM Content Manager passes a document, a folder, or an XPATH query string to the `DKContentOperationsICM` class by using the `VWAttachment` object in the FileNet Business Process Manager Process Designer. The `DKContentOperationsICM` class implements the `IContentOperations` Java interface in IBM Content Manager.

Specify the `VWAttachment` object in the Process Designer in the following format:

```
name|description|attachmenttype|libraryType|libraryID|GUID|VersionID
```

When you design a workflow by using FileNet Business Process Manager with IBM Content Manager, specify the following fields in the `VWAttachment` object:

attachmenttype
Type of attachment. For IBM Content Manager, the **`attachmenttype`** parameter can have the following values:

Document
The value of `attachmenttype` is 3.

```
 VWAttachmentType.ATTACHMENT_TYPE_DOCUMENT
```

You must set the value of `attachmenttype` to 3 for `searchForMany` and `searchForOne` APIs.

Folder The value of `attachmenttype` is 2.

```
VWAttachmentType.ATTACHMENT_TYPE_FOLDER
```

libraryType
For IBM Content Manager, `libraryType` is 4.

```
VWLibraryType.LIBRARY_TYPE_CM
```

libraryID
The name of the IBM Content Manager library server.

GUID The PID string of a document or a folder in IBM Content Manager.

The GUID for the`searchForMany` and `searchForOne` methods is the IBM Content Manager XPATH query string.

Examples of the VWAttachment object

The `VWAttachment` object format for a query is similar to:

```
"||3|4|ICMNLSDB|/NOINDEX[@SOURCE like ""ABC%""]||"
```

The `VWAttachment` object format for a PID string is similar to:

```
"||3|4|icmnlsdb|86 3 ICM8 icmnlsdb7 NOINDEX59 26
  A1001001A07C20A02957G1416018
  A07C20A02957G141601 14 1000|"
```

Related information

Example sample configuration

Attribute mappings for the DKContentOperationsICM class:

When you design a workflow, the IBM Content Manager attribute types are mapped to the CE_Operations data types that are supported by FileNet Business Process Manager.

The IBM Content Manager attributes that are supported by the `DKContentOperationsICM` class are described in the following table.

Table 42. Attribute mappings in the DKContentOperationsICM class

CE_Operations data types	IBM Content Manager attribute types	`DKContentOperationsICM` attribute types
String	Character	String
String	Variable character	String
Integer	Long integer	Integer
Double	Double	Double
java.util.Date	Time stamp IBM Content Manager has milliseconds for the time stamp. When the IBM Content Manager time stamp is converted to a java.util.Date data type, the milliseconds are ignored.	java.util.Date
java.util.Date	Date The time in the java.util.Date data type is ignored because IBM Content Manager keeps only the date for the date attribute type.	java.util.Date
Integer	Short integer	Integer
Double	Decimal	Double

Important: When the data is converted from an integer data type to a short integer type, an exception occurs if there is overflow. The FileNet Business Process Manager workflow process is then added to the exception queue.

Example of converting dates between IBM Content Manager and FileNet Business Process Manager

When the parameters in the getDateProperty() and setDateProperty() methods or the return values of these methods are passed from the FileNet Business Process Manager to IBM Content Manager or from IBM Content Manager to FileNet Business Process Manager, the following date and time stamp data is passed:

java.util.Date

The java.util.Date type in FileNet Business Process Manager is converted to the date type in IBM Content Manager and converted to the java.util.Date type again in FileNet Business Process Manager as described in the following table.

Table 43. Example of date conversion

FileNet Business Process Manager	IBM Content Manager	FileNet Business Process Manager
Mon Jun 16 13:34:59 PST 2008	Jun 16 2008	Mon Jun 16 00:00:00 PST 2008

time stamp

When the java.util.Date type value from FileNet Business Process Manager is passed to IBM Content Manager, the milliseconds are added. When the time stamp from IBM Content Manager is passed to the FileNet Business Process Manager, the milliseconds in the time stamp are ignored.

Table 44. Example of time stamp conversion

FileNet Business Process Manager	IBM Content Manager	FileNet Business Process Manager
Mon Jun 16 13:34:59 PST 2008	2008-06-16-13.34.59.000000	Mon Jun 16 13:34:59 PST 2008

Supported DKContentOperationsICM class methods in IBM Content Manager:

Use the `DKContentOperationsICM` class methods that are supported by IBM Content Manager to design your workflow application.

Important: If versioning is enabled and the attribute values of a document or folder are modified, a new version of that document or folder is created. However, the attachment that contains the PID string is not updated to the new version. If you want to work with the latest version of the document or folder, you must retrieve the latest version.

The methods that are supported by IBM Content Manager in the `DKContentOperationsICM` class are:

getDateProperty

Retrieves a date attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

Parameters

`sourceDocument`

The document or folder from which the date attribute value of the specified attribute is retrieved.

`attrName`

The name of the attribute whose date attribute value is retrieved.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

Return values

Date attribute value.

If a retrieved date value is null in IBM Content Manager, a date (-2000000000000L) value is returned. FileNet Business Process Manager uses this value to indicate a null date. The date (-2000000000000L) is same as Thu Aug 16 12:26:40 PST 1906.

The data type in IBM Content Manager must be a `DKTimestamp` or `DKDate` type to retrieve the date value. If the IBM Content Manager data type is a `DKDate` type, there is no time returned in `java.util.Date` class because the IBM Content Manager date does not contain the time information.

```
Date getDateProperty(VWAttachment sourceDocument, String attrName)
```

getDoubleProperty

Retrieves a double attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be a double or a decimal type. The attribute value is converted from decimal to double for IBM Content Manager decimal data types.

Parameters

sourceDocument
The document or folder from which the double attribute value of the specified attribute is retrieved.

attrName
The name of the attribute whose double attribute value is retrieved.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

Return values
Double attribute value.

If a retrieved double value is null in IBM Content Manager, a double with a 0 value is returned.

```
Double getDoubleProperty(VWAttachment sourceDocument, String attrName)
```

getIntegerProperty
Retrieves an integer attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be of an integer or a short type. The attribute value is converted from short to integer for IBM Content Manager short data types.

Parameters

sourceDocument
The document or folder from which the integer attribute value of the specified attribute is retrieved.

attrName
The name of the attribute whose integer attribute value is retrieved.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

Return values
Integer attribute value.

If a retrieved integer value is null in IBM Content Manager, an integer with a 0 value is returned.

```
Integer getIntegerProperty(VWAttachment sourceDocument, String attrName)
```

getStringProperty
Retrieves a string attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

Parameters

sourceDocument
The document or folder from which the string attribute value of the specified attribute is retrieved.

attrName
The name of the attribute whose string attribute value is retrieved.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

Return values
String attribute value.

If a retrieved string value is null in IBM Content Manager, an empty string is returned.

```
String getStringProperty(VWAttachment sourceDocument, String attrName)
```

getURLFromAttachment

Retrieves the URL of the first document part of the specified document for a document model item type in IBM Content Manager.

Parameters

`sourceDocument`
The document or folder from which the URL of the first document part is retrieved.

`baseURL`
This parameter is not supported by IBM Content Manager.

Return values
If the document is from a resource item type, the URL for the resource item is returned. If the document does not have any parts or if the document is from a nonresource item type, an empty string is returned.

```
String getURLFromAttachment(VWAttachment sourceDocument, String baseURL)
```

getURLFromAttachmentWithIndex

Retrieve the URL of the content based on an index of the total content objects for the specified part type of the specified document.

Parameters

`sourceDocument`
The document or folder from which the URL is retrieved.

`partType`
IBM Content Manager part type for the document model document. For example, ICMBASE, ICMBASETEXT, ICMANNOTATION, ICMNOTELOG, or ICMBASESTREAM. If the document is from a resource item type, set the part type to an empty string and set the index to 1. If the document is from a non resource item type, set the part type to an empty string and set the index to 0.

`index`
Index of the content objects for the specified part type. This value cannot exceed the total number of part content objects for the specified part type. If the document has no parts for that part type, specify 0.

Return values
The URL of the document part. If the document is from a document model item type, the URL of the part content of a document will be returned based on the part type and index. If the document is from a nonresource item type, an empty string will be returned.

```
String getURLFromAttachmentWithIndex(VWAttachment sourceDocument,
  String partType, int index)
```

setDateProperty

Sets a date attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

Parameters

destDocument
The document or folder that contains the specified attribute.

attrName
The name of the attribute whose date attribute value is set.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

val
Date attribute value.

Return values
Date attribute value.

The data type in IBM Content Manager must be a `DKTimestamp` or `DKDate` type to set the date value. If the IBM Content Manager data type is a `DKDate` type, no time is returned in the `java.util.Date` class because the IBM Content Manager date does not contain the time information.

```
Date setDateProperty(VWAttachment destDocument, String attrName, Date val)
```

setDoubleProperty
Sets a double attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be a double or decimal type. The attribute value is converted from double to decimal for IBM Content Manager decimal data types.

Parameters

destDocument
The document or folder that contains the specified attribute.

attrName
The name of the attribute whose double attribute value is set.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

val
Double attribute value.

```
void setDoubleProperty(VWAttachment destDocument,
  String attrName, Double val)
```

setIntegerProperty
Sets an integer attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be an integer or short type. The attribute value is converted from integer to short for short data types in IBM Content Manager.

Parameters

destDocument
The document or folder that contains the specified attribute.

attrName
The name of the attribute whose integer attribute value is set.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

val
Integer attribute value.

```
void setIntegerProperty(VWAttachment destDocument,
  String attrName, Integer val)
```

setStringProperty
Sets a string attribute value for the specified attribute name from the specified document or folder in IBM Content Manager.

Parameters

destDocument
The document or folder that contains the specified attribute.

attrName
The name of the attribute whose string attribute value is set.

The attribute name is specified in the following format: Attribute Name or Attribute Group.Attribute Name.

val
String attribute value.

```
void setStringProperty(VWAttachment destDocument,
  String attrName, String val)
```

file Adds a document to the specified folder in IBM Content Manager.

Parameters

sourceDocument
The document that is added.

destFolder
The folder to which the document is added.

```
void file(VWAttachment sourceDocument, VWAttachment destFolder)
```

unfile Removes a document from the specified folder in IBM Content Manager.

Parameters

sourceDocument
The document that is removed.

destFolder
The folder from which the document is removed.

```
void unfile(VWAttachment sourceDocument, VWAttachment destFolder)
```

checkin
Updates and checks in a document in IBM Content Manager. Only first-level children are supported.

Important: The document can be a resource item that has content. If the document is a nonresource item, it must not have any content or MIME type information.

Parameters

sourceDocument
The document that is checked in.

propArray
Contains the item type name and the attributes for the item type. The array must be in the following format:

```
{itemTypeName,
  "DOCPARTTYPE" (This string represents a document
  model item type part type), documentPartType,
   {"ITATTR" (This string represents an item type
   attribute), baseAttrName1, dataType1, baseAttrValue1,
   "ITATTR", baseAttrName2, dataType2, baseAttrValue2,
    and so on }
     {"CCATTR" (string represents a child component
      attribute), childComponentName1, childAttrName1,
      dataType1, numberOfValues1, {childAttrValue1,
    childAttrValue2, and so on},
      "CCATTR", childComponentName2, childAttrName2,
      dataType2, numberOfValues2,
   {childAttrValue1, childAttrValue2, and so on
   }
  }
}
```

The possible data types are:

Input Type
IBM Content Manager data type.

TIMESTAMP
DKTimestamp. The time stamp format is: MM/dd/yyyy HH:mm:ss.

DATE DKDate. The date format is: MM/dd/yyyy.

DOUBLE
Double.

INTEGER
Integer.

STRING
String.

SHORT
Short.

DECIMAL
java.math.BigDecimal.

Important:
- If you want to set an attribute for an item type or child component to null, do not specify the ITATTR or CCATTR section for that attribute.
- If an attribute belongs to an attribute group, the attribute must be in the following format for the ITATTR or CCATTR section: *Attribute Group Name.Attribute Name*.
- If the item type is a document model item type and you specify the MIME type and content, you must specify the DOCPARTTYPE keyword and value.
- If the item type is a resource item type, do not specify the DOCPARTTYPE keyword and value.
- Using child components is optional and depends on the item type definition. You can use child components for multi-valued attributes.

Example of propArray with NOINDEX item type

The following example shows the values of elements in the propArray string array:

```
String[0] NOINDEX
String[1] DOCPARTTYPE
String[2] ICMBASE
String[3] ITATTR
String[4] SOURCE
String[5] STRING
String[6] Source1
String[7] ITATTR
String[8] USER_ID
String[9] STRING
String[10] Robert
String[11] ITATTR
String[12] TIMESTAMP
String[13] TIMESTAMP
String[14] 03/26/2008 13:38:00
```

Example of a propArray with the BOOK item type having a child component named AUTHORS

The following example shows the values of elements in the propArray string array:

```
String array
String[0] BOOK
String[1] ITATTR
String[2] TITLE
String[3] STRING
String[4] The Blue Dragon
String[5] CCATTR
String[6] AUTHORS
String[7] NAME
String[8] STRING
String[9] 2
String[10] Robert Nelson
String[11] Tom Harrison
```

`fileName`

The name of the file. This parameter is not supported by IBM Content Manager.

`mimeType`

The MIME type of the content.

`content`

The string that represents the content.

Return values

An attachment to the checked in document or folder.

```
VWAttachment checkin(VWAttachment sourceDocument,
  String[] propArray, String fileName, String mimeType, String content)
```

checkout

Checks out a document from IBM Content Manager.

Parameters

`sourceDocument`

The document that is checked out.

```
void checkout(VWAttachment sourceDocument)
```

copy Makes a copy of the specified document and adds this new document to IBM Content Manager.

Parameters

sourceDocument
The document that is copied.

file
Set to True if the new document copy will be placed in the same folder as the specified document. If the specified document belongs to multiple folders, the document copy is added to the same folders. If the specified document does not belong to any folder and the **file** parameter is set to True, the new document copy is not added to any folder. If the **file** parameter is set to False, the new document copy is not added to the same folders as the specified document.

Return values
An attachment to the new document copy.

```
VWAttachment copy(VWAttachment sourceDocument, boolean file)
```

copyMany
Makes a copy of each specified document in the array and adds this new document to IBM Content Manager.

Parameters

sourceDocuments
An array of documents that are copied.

file
Set to True if each new document copy will be placed in the same folder as each specified document. If the specified document belongs to multiple folders, the document copy is added to the same folders. If the specified document does not belong to any folder and the **file** parameter is set to True, the new document copy is not added to any folder. If the **file** parameter is set to False, the new document copy is not added to the same folders as the specified document.

Return values
An array of attachments.

```
VWAttachment[] copyMany(VWAttachment[] sourceDocuments, boolean file)
```

createDocument
Creates a document with a part and adds this document to a specified folder in IBM Content Manager. Only first-level children are supported.

Important: The user must create a folder before creating a document. The document can be a resource item that has content. If the document is a nonresource item, it must not have any content or MIME type information.

Parameters

destFolder
The folder where the newly created document is added.

className
This parameter is not used by IBM Content Manager.

propArray
Contains the library server name, the item type, and the attributes for the item type. The array must be in the following format:

```
{itemTypeName,
  "DOCPARTTYPE" (This string represents a document
  model item type part type), documentPartType,
   {"ITATTR" (This string represents an item type
   attribute), baseAttrName1, dataType1, baseAttrValue1,
   "ITATTR", baseAttrName2, dataType2, baseAttrValue2,
    and so on }
     {"CCATTR" (string represents a child component
      attribute), childComponentName1, childAttrName1,
      dataType1, numberOfValues1, {childAttrValue1,
    childAttrValue2, and so on},
      "CCATTR", childComponentName2, childAttrName2,
      dataType2, numberOfValues2,
   {childAttrValue1, childAttrValue2, and so on
   }
  }
}
```

The possible data types are:

Input Type
IBM Content Manager data type.

TIMESTAMP
DKTimestamp. The time stamp format is: MM/dd/yyyy HH:mm:ss.

DATE DKDate. The date format is: MM/dd/yyyy.

DOUBLE
Double.

INTEGER
Integer.

STRING
String.

SHORT
Short.

DECIMAL
java.math.BigDecimal.

Important:

- If you want to set an attribute for an item type or child component to null, do not specify the ITATTR or CCATTR section for that attribute.
- If an attribute belongs to an attribute group, the attribute must be in the following format for the ITATTR or CCATTR section:

  Attribute Group Name.*Attribute Name*
- If the item type is a document model item type and you specify the MIME type and content, you must specify the DOCPARTTYPE keyword and value.
- If the item type is a resource item type, do not specify the DOCPARTTYPE keyword and value.
- Using child components is optional and depends on the item type definition. You can use child components for multi-valued attributes.

Example of propArray with NOINDEX item type

The following example shows the values of elements of the propArray string array:

```
String[0] NOINDEX
String[1] DOCPARTTYPE
String[2] ICMBASE
String[3] ITATTR
String[4] SOURCE
String[5] STRING
String[6] Source1
String[7] ITATTR
String[8] USER_ID
String[9] STRING
String[10] Robert
String[11] ITATTR
String[12] TIMESTAMP
String[13] TIMESTAMP
String[14] 03/26/2008 13:38:00
```

Example of a propArray with the BOOK item type having a child component named AUTHORS

The following example shows the values of elements of the propArray string array:

```
String array
String[0] BOOK
String[1] ITATTR
String[2] TITLE
String[3] STRING
String[4] The Blue Dragon
String[5] CCATTR
String[6] AUTHORS
String[7] NAME
String[8] STRING
String[9] 2
String[10] Robert Nelson
String[11] Tom Harrison
```

`fileName`
The name of the file. This parameter is not supported by IBM Content Manager.

`mimeType`
The MIME type of the content. If the item or the document does not have any content or is a nonresource item, do not specify the **`mimeType`** parameter.

`content`
The string that represents the content for a part or resource item. If the document or item does not have any content, specify an empty string for the **`content`** parameter.

Return values
An attachment to the created document.

```
VWAttachment createDocument(VWAttachment destFolder,
  String className, String[] propArray, String fileName,
  String mimeType, String content)
```

createFolder
Creates a folder in IBM Content Manager. This folder can be a parent folder or a subfolder. Only first-level children are supported.

Parameters

`parentFolder`
The parent folder that is created.

If the folder that is created does not have a parent folder, set the ID of the VWAttachment to empty string. For example, the VWAttachment is similar to: `"||2|4|icmnlsdb||"`.

`className`
The folder class name. This parameter is not supported by IBM Content Manager.

`propArray`
Contains the library server name, the item type, and the attributes for the item type. The array must be in the following format:

```
{DSNAME, libraryServer, itemTypeName,
  {"ITATTR" (This string represents an item type
   attribute), baseAttrName1, dataType1, baseAttrValue1,
   "ITATTR", baseAttrName2, dataType2, baseAttrValue2,
    and so on }
     {"CCATTR" (string represents a child component
      attribute), childComponentName1, childAttrName1,
      dataType1, numberOfValues1, {childAttrValue1,
    childAttrValue2, and so on},
      "CCATTR", childComponentName2, childAttrName2,
      dataType2, numberOfValues2,
   {childAttrValue1, childAttrValue2, and so on
   }
  }
}
```

The possible data types are:

Input Type
IBM Content Manager data type.

TIMESTAMP
DKTimestamp. The time stamp format is: MM/dd/yyyy HH:mm:ss.

DATE DKDate. The date format is: MM/dd/yyyy.

DOUBLE
Double.

INTEGER
Integer.

STRING
String.

SHORT
Short.

DECIMAL
java.math.BigDecimal.

Important:

- If you want to set an attribute for an item type or child component to null, do not specify the ITATTR or CCATTR section for that attribute.

- If an attribute belongs to an attribute group, the attribute must be in the following format for the ITATTR or CCATTR section: *Attribute Group Name.Attribute Name*.
- If the item type is a resource item type, do not specify the DOCPARTTYPE keyword and value.
- Using child components is optional and depends on the item type definition. You can use child components for multi-valued attributes.

Example of propArray with NOINDEX item type

The following example shows the values of elements of the propArray string array:

```
String[0] DSNAME
String[1] ICMNLSDB
String[2] NOINDEX
String[3] ITATTR
String[4] SOURCE
String[5] STRING
String[6] Source1
String[7] ITATTR
String[8] USER_ID
String[9] STRING
String[10] Robert
String[11] ITATTR
String[12] TIMESTAMP
String[13] TIMESTAMP
String[14] 03/26/2008 13:38:00
```

Example of a propArray with BOOK item type having a child component named AUTHORS

The following example shows the values of elements of the propArray string array:

```
String array
String[0] DSNAME
String[1] ICMNLSDB
String[2] BOOK
String[3] ITATTR
String[4] TITLE
String[5] STRING
String[6] The Blue Dragon
String[7] CCATTR
String[8] AUTHORS
String[9] NAME
String[10] STRING
String[11] 2
String[12] Robert Nelson
String[13] Tom Harrison
```

Return values

An attachment to the created folder.

```
VWAttachment createFolder(VWAttachment parentFolder,
  String className, String[] propArray)
```

delete Deletes the specified document or folder in IBM Content Manager.

Parameters

`sourceDocument`

The document that is to be deleted.

```
void delete(VWAttachment sourceDocument)
```

deleteMany

Deletes an array of specified documents or folders in IBM Content Manager.

Parameters

`sourceDocuments`
An array of documents that are to be deleted.

```
void delete(VWAttachment[] sourceDocuments)
```

fileMany

Adds one or more documents to the specified folder in IBM Content Manager.

Parameters

`folder`
The folder to which the documents are added.

`sourceDocuments`
An array of documents that are added.

```
void fileMany(VWAttachment folder, VWAttachment[] sourceDocuments)
```

getContent

Retrieves the part content of a document for a document model item type in IBM Content Manager.

Parameters

`sourceDocument`
The document from which the part content is retrieved.

Return values

A String array that contains three elements:

1. File name, which will always be an empty string.
2. One of the following MIME types: text/plain, text/css, text/html, or text/xml.
3. Content of the part.

If the document has more than one part, only the content for the first part will be returned. If the document has no content, three empty strings will be returned in the string array.

```
String[] getContent(VWAttachment sourceDocument)
```

getContentWithIndex

Retrieves content based on an index of the total content objects for the specified part type of the specified document.

Parameters

`sourceDocument`
The document from which the part content is retrieved.

`partType`
IBM Content Manager part type for the document model document. For example, ICMBASE, ICMBASETEXT, ICMANNOTATION, ICMNOTELOG, or ICMBASESTREAM. If the document is from a resource item type, then set the part type to an empty string and set the index to 1. If the document is from a non resource item type, then set the part type to an empty string and set the index to 0.

index
Index of the content objects for the specified part type. This value cannot exceed the total number of part content objects for the specified part type. If the document has no parts for that part type, specify 0.

Return values
A String array that contains three elements:
1. File name, which will always be an empty string.
2. One of the following MIME types: text/plain, text/css, text/html, or text/xml.
3. Content of the part.

If the document is from a document model item type, the part content of a document will be returned based on the content type and index. If the document has no parts for that part type, three empty strings will be returned.

```
String[] getContentWithIndex(VWAttachment sourceDocument,
  String partType, int index)
```

getContentCount
Retrieves the number of content objects for a specified part type.

Parameters

sourceDocument
The document or folder from which the number of content objects is retrieved.

partType
IBM Content Manager part type for the document model document. If the document is from a resource item type or a non resource item type, then set the part type to an empty string.

Return values

1	The specified document is from a resource item type.
0	The specified document is from a document model item type wherein no content is found for the specified part type, such as a non resource item type.

```
int getContentCount(VWAttachment sourceDocument, String partType)
```

getLatestAttachmentVersion
Retrieves the latest attachment version of document or folder in IBM Content Manager.

Parameters

sourceDocument
The document or folder from which the latest attachment version is retrieved.

Return values
A VWAttachment (the latest attachment version).

```
VWAttachment getLatestAttachmentVersion(VWAttachment sourceDocument)
```

getMultiChildDateProperty
Retrieves a list of date attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

sourceDocument
The document or folder from which the date attribute value of the specified attribute is retrieved.

attrName
The name of the attribute whose date attribute value is retrieved.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

childAttrNames
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrTypes
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrValues
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

Return values
An array of date attribute values.

If a retrieved date value is null in IBM Content Manager, a date (-2000000000000L) value is returned. FileNet Business Process Manager uses this value to indicate a null date. The date (-2000000000000L) is equivalent to Thu Aug 16 12:26:40 PST 1906.

The data type in IBM Content Manager must be a `DKTimestamp` or `DKDate` type to retrieve the date value. If the IBM Content Manager data type is a `DKDate` type, there is no time component returned in the `java.util.Date` class because the IBM Content Manager date does not contain the time information.

```
Date[] getMultiChildDateProperty(VWAttachment sourceDocument,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues)
```

getMultiChildDoubleProperty
Retrieves a list of double attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be of double or decimal type. The attribute value is converted from decimal to double for decimal data type in IBM Content Manager.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

sourceDocument
The document or folder from which the double attribute value of the specified attribute is retrieved.

attrName
The name of the attribute whose double attribute value is retrieved.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

childAttrNames
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrTypes
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrValues
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

Return values
An array of double attribute values.

If a retrieved double value is null in IBM Content Manager, a double with a 0 value is returned.

```
Double[] getMultiChildDoubleProperty(VWAttachment sourceDocument,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues)
```

getMultiChildIntegerProperty
Retrieves a list of integer attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be of an integer or a short type. The attribute value is converted from short to integer for IBM Content Manager short data types.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

sourceDocument
The document or folder from which the integer attribute value of the specified attribute is retrieved.

attrName
The name of the attribute whose integer attribute value is retrieved.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

childAttrNames
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrTypes
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrValues
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

Return values
An array of integer attribute values.

If a retrieved integer value is null in IBM Content Manager, an integer with a 0 value is returned.

```
Integer[] getMultiChildIntegerProperty(VWAttachment sourceDocument,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues)
```

getMultiChildStringProperty
Retrieves a list of string attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

sourceDocument
The document or folder from which the string attribute value of the specified attribute is retrieved.

attrName
The name of the attribute whose string attribute value is retrieved.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

childAttrNames
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrTypes
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

childAttrValues
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

Return values
An array of string attribute values.

If a retrieved string value is null in IBM Content Manager, an empty string is returned.

```
String[] getMultiChildStringProperty(VWAttachment sourceDocument,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues)
```

getMultiDateProperty

Retrieves a list of date attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The data type in IBM Content Manager must be a `DKTimestamp` or `DKDate` type to retrieve the date value. If the IBM Content Manager data type is a `DKDate` type, no time is returned in the `java.util.Date` class because the IBM Content Manager date does not contain the time information.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

`sourceDocument`
The document or folder from which a list of date attribute values of the specified attribute is retrieved.

`attrName`
The name of the attribute whose date attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

Return values
An array of date attribute values.

```
Date[] getMultiDateProperty(VWAttachment sourceDocument,
  String attrName)
```

getMultiDoubleProperty

Retrieves a list of double attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be a double or a decimal type. The attribute value is converted from a decimal type to a double type for IBM Content Manager decimal data types.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

`sourceDocument`
The document or folder from which the list of double attribute values of the specified attribute is retrieved.

`attrName`
The name of the attribute whose double attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

Return values
An array of double attribute values.

```
Double[] getMultiDoubleProperty(VWAttachment sourceDocument,
  String attrName)
```

getMultiIntegerProperty

Retrieves a list of integer attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be an integer or a short type. The attribute value is converted from short to integer for IBM Content Manager short data types.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

`sourceDocument`
The document or folder from which the list of integer attribute values of the specified attribute is retrieved.

`attrName`
The name of the attribute whose integer attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

Return values
An array of integer attribute values.

```
Integer[] getMultiIntegerProperty(VWAttachment sourceDocument,
  String attrName)
```

getMultiStringProperty

Retrieves a list of string attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

Important: IBM Content Manager cannot guarantee the order of multiple property values returned by this method.

Parameters

`sourceDocument`
The document or folder from which the list of double attribute values of the specified attribute is retrieved.

`attrName`
The name of the attribute whose string attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

Return values
An array of string attribute values.

If a retrieved string value is null in IBM Content Manager, an empty string is returned.

```
String[] getMultiStringProperty(VWAttachment sourceDocument,
  String attrName)
```

move Removes a document from a folder and adds it to another folder in IBM Content Manager.

Parameters

`folder`
The folder to which the document is added.

sourceDocument
The document that is moved.

```
void move(VWAttachment folder, VWAttachment sourceDocument)
```

moveMany
Removes one or more documents from their folders, adds the specified documents to another folder, and saves that folder in IBM Content Manager.

Parameters

folder
The folder to which the documents are added.

sourceDocuments
An array of documents that is moved.

```
void moveMany(VWAttachment folder, VWAttachment[] sourceDocuments)
```

searchForOne
Searches for a document or folder in IBM Content Manager. This method uses a query that is specified in the VWAttachment object.

Parameters

search
The VWAttachment object that specifies the search. The following fields in the VWAttachment object must be set:

attachmenttype
The type of the attachment. For IBM Content Manager, this field has the following values, depending on whether a document or a folder is specified:

- Document search: The `attachmenttype` field is set to 3.
- Folder search: The `attachmenttype` field is set to 2.

libraryType
For IBM Content Manager, the `librarytype` field is set to `VWLibraryType.LIBRARY_TYPE_CM`, which is equal to 4.

libraryID
The name of the IBM Content Manager library server.

GUID

The IBM Content Manager XPATH query string.

objectType
Type of document or folder. This parameter is not supported by IBM Content Manager.

itemIds
An array of values specifying the item IDs from the search template.

values
An array of values that are correlated to the array of item IDs. This parameter is not supported.

Return values
A VWAttachment object that represents the first document or folder matched in the search. If a match is not found, an empty VWAttachment object is returned.

```
VWAttachment searchForOne(VWAttachment search,
  String objectType, int[] itemIds, String[] values)
```

searchForMany

Searches for one or more documents or folders in IBM Content Manager. The query for this method is specified in a VWAttachment object. For each library server, the default value for the maximum results that are returned is 50, which can be modified by specifying the `queryMaxResultsList` option for the IBM Content Manager login module.

Parameters

`search`
The VWAttachment object that specifies the search. The following fields in the VWAttachment object must be set:

attachmenttype
The type of the attachment. For IBM Content Manager, this field has the following values, depending on whether a document or a folder is specified:

- Document search: The `attachmenttype` field is set to 3.
- Folder search: The `attachmenttype` field is set to 2.

libraryType
For IBM Content Manager, the `librarytype` field is set to`VWLibraryType.LIBRARY_TYPE_CM`, which is equal to 4.

libraryID
The name of the IBM Content Manager library server.

GUID

The IBM Content Manager XPATH query string.

`objectType`
Type of document or folder. This parameter is not supported by IBM Content Manager.

`itemIds`
An array of values specifying the item IDs from the search template.

`values`
An array of values that are correlated to the array of item IDs. This parameter is not supported by IBM Content Manager.

Return values
An array of VWAttachment objects that represent the documents and folders that match the search. If a match is not found, an empty VWAttachment array is returned.

```
VWAttachment[] searchForMany(VWAttachment search,
  String objectType, int[] itemIds, String[] values)
```

setMultiChildDateProperty

Sets a list of date attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The data type in IBM Content Manager must be a `DKTimestamp` or `DKDate` type to set the date value. If the IBM Content Manager data type is a `DKDate` type, there is no time returned in `java.util.Date` class because the IBM Content Manager date does not contain the time information.

For a child component with a single attribute, you can supply any number of child attribute values.

For a child component with multiple attributes:

- If the item does not have any children specified for the child component type, then you can specify any number of multi value attributes which will cause the same number of children to be created for the child component.
- If the item already has a certain number of children specified for the child component type, then you can only specify the same number of attribute values as the number of children. If the user specifies a number of attribute values that differ from the number of children, then an exception will be thrown.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attribute whose date attribute value is set.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

`childAttrNames`
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrTypes`
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrValues`
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`values`
An array of date attribute values.

```
setMultiChildDateProperty(VWAttachment sourceDocument,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues, Date[] values)
```

setMultiChildDoubleProperty

Sets a list of double attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager may be of type double or decimal. The attribute value is converted from double to decimal for IBM Content Manager decimal data types.

For a child component with a single attribute, you can supply any number of child attribute values.

For a child component with multiple attributes:

- If the item does not have any children specified for the child component type, then you can specify any number of multi value attributes which will cause the same number of children to be created for the child component.
- If the item already has a certain number of children specified for the child component type, then you can only specify the same number of attribute values as the number of children. If the user specifies a number of attribute values that differ from the number of children, then an exception will be thrown.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attribute whose double attribute value is set.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

`childAttrNames`
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrTypes`
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrValues`
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`values`
An array of double attribute values.

```
void setMultiChildDoubleProperty(VWAttachment sourceDocument,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues, Double[] values)
```

setMultiChildIntegerProperty

Sets a list of integer attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be an integer or short type. The attribute value is converted from integer to short for IBM Content Manager short data types.

For a child component with a single attribute, you can supply any number of child attribute values.

For a child component with multiple attributes:

- If the item does not have any children specified for the child component type, then you can specify any number of multi value attributes which will cause the same number of children to be created for the child component.
- If the item already has a certain number of children specified for the child component type, then you can only specify the same number of attribute values as the number of children. If the user specifies a number of attribute values that differ from the number of children, then an exception will be thrown.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attribute whose integer attribute value is set.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

`childAttrNames`
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrTypes`
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrValues`
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`values`
An array of integer attribute values.

```
void setMultiChildIntegerProperty(VWAttachment sourceDocumentt,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues, Integer[] values)
```

setMultiChildStringProperty

Sets a list of string attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

For a child component with a single attribute, you can supply any number of child attribute values.

For a child component with multiple attributes:

- If the item does not have any children specified for the child component type, then you can specify any number of multi value attributes which will cause the same number of children to be created for the child component.
- If the item already has a certain number of children specified for the child component type, then you can only specify the same number of

attribute values as the number of children. If the user specifies a number of attribute values that differ from the number of children, then an exception will be thrown.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attribute whose string attribute value is set.

The attribute name is specified in the following format: ChildComponentName1/ChildComponentName2/.../attributeName or ChildComponentName1/ChildComponentName2/.../attributeGroup.attributeName.

`childAttrNames`
An array of child attribute names. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrTypes`
An array of child attribute types. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`childAttrValues`
An array of child attribute values. The number of elements in the arrays specified for childAttrNames, childAttrTypes and childAttrValues must be the same.

`values`
An array of string attribute values.

```
void setMultiChildStringProperty(VWAttachment sourceDocument,
  String attrName, String[] childAttrNames, String[] childAttrTypes,
  String[] childAttrValues, String[] values)
```

setMultiDateProperty

Sets a list of date attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The data type in IBM Content Manager must be a `DKTimestamp` or `DKDate` type to set the date value. If the IBM Content Manager data type is a `DKDate` type, no time is returned in the `java.util.Date` class because the IBM Content Manager date does not contain the time information.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attribute whose date attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

values
An array of date attribute values.

```
void setMultiDateProperty(VWAttachment sourceDocument,
  String attrName, Date[] values)
```

setMultiDoubleProperty
Sets a list of double attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be a double or a decimal type. The attribute value is converted from double to decimal for IBM Content Manager decimal data types.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

sourceDocument
The document or folder that contains the specified attribute.

attrName
The name of the attributes whose double attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

values
An array of double attribute values.

```
void setMultiDoubleProperty(VWAttachment sourceDocument,
  String attrName, Double[] values)
```

setMultiIntegerProperty
Sets a list of integer attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

The attribute value in IBM Content Manager can be an integer or a short type. The attribute value is converted from integer to short for IBM Content Manager short data types.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

sourceDocument
The document or folder that contains the specified attribute.

attrName
The name of the attributes whose integer attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

values
An array of integer attribute values.

```
void setMultiIntegerProperty(VWAttachment sourceDocument,
  String attrName, Integer[] values)
```

setMultiStringProperty
Sets a list of string attribute values for the specified attribute name from the specified document or folder in IBM Content Manager.

Important: IBM Content Manager cannot guarantee the order of multiple property values set by this method.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attributes whose string attribute values are retrieved. The attribute names are in the form of `<ChildComponentName/attributeName>` or `<ChildComponentName/attributeGroup.attributeName>`.

`values`
An array of string attribute values.

```
void setMultiStringProperty(VWAttachment sourceDocument,
  String attrName, String[] values)
```

Unsupported DKContentOperationsICM class methods in IBM Content Manager:

The `DKContentOperationsICM` class methods that are not supported by IBM Content Manager cannot be used to design a workflow application.

The methods that are not supported by the `DKContentOperationsICM` class are:

cancelCheckout
Cancels a checkout.

Parameters

`sourceDocument`
The document on which to cancel the checkout.

```
void cancelCheckout(VWAttachment sourceDocument)
```

getBooleanProperty
Retrieves a Boolean attribute value for the specified attribute name from the specified document or folder in IBM Content Manager. This method is not supported because IBM Content Manager does not support Boolean attribute types.

Parameters

`sourceDocument`
The document or folder from which the Boolean attribute value of the specified attribute is retrieved.

`attrName`
The name of the attribute whose date attribute value is retrieved.

Return values
Boolean attribute value.

```
Boolean getBooleanProperty(VWAttachment sourceDocument, String attrName)
```

setBooleanProperty
Sets a Boolean attribute value for the specified attribute name from the specified document or folder in IBM Content Manager. This method is not supported because IBM Content Manager does not support Boolean attribute types.

Parameters

destDocument
The document or folder that contains the specified attribute.

attrName
The name of the attribute whose date attribute value is set.

val
The Boolean attribute value to set.

```
void setBooleanProperty(VWAttachment destDocument,
String attrName, Boolean val)
```

publish
Publishes a source document by using the specified publishing template. This method is not supported because IBM Content Manager does not support the function.

Parameters

sourceDocument
The document or folder to publish.

publishing template
The publishing template used for publishing the source document.

```
VWAttachment publish(VWAttachment sourceDocument,
VWAttachment publishingTemplate)
```

applySecurityTemplate
Applies a security template to a specified document or folder. This method is not supported because IBM Content Manager does not support the function.

Parameters

sourceDocument
The documents or folders to which the security template is applied.

templateGUID
The security template to apply.

```
void applySecurityTemplate(VWAttachment sourceDocuments,
String templateGUID)
```

applySecurityTemplateMany
Applies a security template to one or more specified documents or folders. This method is not supported because IBM Content Manager does not support the function.

Parameters

sourceDocuments
An array of documents or folders to which the security template is applied.

templateGUID
The security template to apply.

```
void applySecurityTemplateMany(VWAttachment[] sourceDocument,
String templateGUID)
```

changeClass
Changes the class of a document or a folder in IBM Content Manager. This method is not supported because IBM Content Manager does not support the function.

Parameters

`sourceDocuments`
The document or folder whose class is changed.

`classId`
The new class of the document or folder.

```
void changeClass(VWAttachment sourceDocument, String classId)
```

createCustomObject
Creates a custom object for the specified class. The custom object is added to a specified folder. This method is not supported because IBM Content Manager does not support the function.

Parameters

`destFolder`
The folder where the object is created.

`className`
The custom object class name.

`propArray`
An array of property values to assign to the new custom object. This array must be in the following format:

```
{itemTypeName,
  {"ITATTR" (This string represents an item type
   attribute), baseAttrName1, dataType1, baseAttrValue1,
   "ITATTR", baseAttrName2, dataType2, baseAttrValue2,
    and so on }
     {"CCATTR" (string represents a child component
      attribute), childComponentName1, childAttrName1,
      dataType1, numberOfValues1, {childAttrValue1,
    childAttrValue2, and so on},
      "CCATTR", childComponentName2, childAttrName2,
      dataType2, numberOfValues2,
   {childAttrValue1, childAttrValue2, and so on
   }
  }
}
```

The possible data types are:

Input Type
IBM Content Manager data type.

TIMESTAMP
DKTimestamp. The time stamp format: MM/dd/yyyy HH:mm:ss.

DATE DKDate. The date format: MM/dd/yyyy.

DOUBLE
Double.

INTEGER
Integer.

STRING
String.

SHORT
Short.

DECIMAL
java.math.BigDecimal.

Important:

- If you want to set an attribute for an item type or child component to null, do not specify the ITATTR or CCATTR section for that attribute.
- If an attribute belongs to an attribute group, the attribute must be in the following format for the ITATTR or CCATTR section: *Attribute Group Name.Attribute Name*.
- If the item type is a document model item type and you have specified the MIME type and content, you must specify the DOCPARTTYPE keyword and value.
- Using child components is optional and depends on the item type definition. You can use child components for multi-valued attributes.

Return values

An attachment to the created document.

```
VWAttachment createCustomObject(VWAttachment destFolder,
String className, String[] propArray)
```

getObjFromPath

Retrieves a custom object, folder, or document from IBM Content Manager by using the specified path. This method is not supported because IBM Content Manager does not support the function.

Parameters

osName
The name of the library server.

path
The object path from which the document or folder is retrieved. This parameter is not supported by IBM Content Manager.

objType
The object type of the document or folder. This parameter is not supported by IBM Content Manager.

```
VWAttachment getObjFromPath(String osName, String path,
String objType)
```

getObjectProperty

Retrieves a object attribute value for the specified attribute name from the specified document or folder in IBM Content Manager. This operation is not supported because IBM Content Manager does not have object attribute types.

Parameters

sourceDocument
The document or folder that contains the specified attribute.

attrName
The name of the attribute whose object attribute value is retrieved.

Return values

The object attribute value.

```
VWAttachment getObjectProperty(VWAttachment sourceDocument,
String attrName)
```

getMultiBooleanProperty

Retrieves a list of Boolean attribute values for the specified attribute name from the specified document or folder in IBM Content Manager. This method is not supported because IBM Content Manager does not support boolean attribute types.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attributes whose Boolean attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>`.

Return values
An array of Boolean attribute values.

```
Boolean[] getMultiBooleanProperty(VWAttachment sourceDocument,
String attrName)
```

getMultiObjectProperty

Retrieves a list of object attribute values for the specified attribute name from the specified document or folder in IBM Content Manager. This operation is not supported because IBM Content Manager does not have object attribute types.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attributes whose object attribute values are retrieved. The names are in the form of `<ChildComponentName/attributeName>`.

Return values
An array of VWAttachment attribute values.

```
VWAttachment[] getMultiObjectProperty(VWAttachment sourceDocument,
String attrName)
```

setMultiBooleanProperty

Sets a list of Boolean attribute values for a specified attribute name from the specified document or folder in IBM Content Manager. This method is not supported because IBM Content Manager does not support Boolean attribute types.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attribute whose Boolean attribute value is set. The names are in the form of `<ChildComponentName/attributeName>`

`values`
An array of Boolean attribute values to set.

```
void setMultiBooleanProperty(VWAttachment sourceDocument,
String attrName, Boolean[] values)
```

setObjectProperty

Retrieves an object attribute value for specified attribute name from the specified document or folder in IBM Content Manager. This operation is not supported because IBM Content Manager does not have object attribute types.

Parameters

`sourceDocument`
The document or folder that contains the specified attribute.

`attrName`
The name of the attribute whose VWAttachment attribute value is set.

`attVal`
An array of VWAttachment attribute values to set.

```
setObjectProperty(VWAttachment sourceDocument,
String attrName, VWAttachment attVal)
```

Creating a FileNet Business Process Manager workflow definition by using the CE_Operations component

To create a workflow definition, start with an empty workflow and add the individual steps in a process. Then, specify the participants, fields, attachments and other resources for each step, and specify the routes that determine the steps.

Important: If the workflow definition document contains one workflow definition, the default file format is PEP. If the workflow definition document contains more than one workflow definition, the default file format is XPDL Version 2.0. If you saved a workflow collection in XPDL format, you cannot then save it in PEP format, even if it contains only one workflow definition.

For information on which format to use when saving your workflow definition, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Getting started** > **Save a workflow definition**.

Creating a workflow definition by using the Process Designer:

Start with a empty workflow that contains the default settings of FileNet Business Process Manager.

For information on how to start your workflow, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Getting started** > **Start a new workflow definition**.

To create an empty workflow definition:

1. Open a browser and enter the following Web address: `http://<Application Engine name:portnumber>/Workplace`.
2. In the Workplace client, click the **Author** tab.
3. On the Author page, click **Process Designer**.
4. From the Workplace Process Designer window, click **File** > **New Workflow**. A new workflow collection is created with an empty workflow definition. The workflow map displays the Launch step, which is the first step in any workflow.

To define the workflow, add component steps.

Adding a component step:

Add a component step on the workflow map to represent each activity in the workflow. The component step uses the CE_Operations component queue to manage documents and attachments in the IBM Content Manager repository.

To add a step to the workflow, you must open the Process Designer by using a Workplace client and add a launch step.

For information about how to use Process Designer to add a step, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Steps** > **Add a step**.

To add a step to the workflow:

1. In the Step Palette section, select **BPM Palette** from the **View Palette** list. The type of steps that you want to use are displayed in the Step Palette area.

   For documentation on how to use a Step Palette, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Steps** > **Use step palettes**.
2. Drag the **Component** step from the Step Palette to the workflow map area.

   The component step routes work to operations in custom JMS components. In the workflow, the component step connects to a component queue configured for one or more operations in the external component.

   For information about the component step, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Steps** > **Component steps**.
3. In the Properties pane, replace the default step name in the **Name** field with a meaningful name.
4. Click the **General** tab in the Properties pane. For each step in the workflow, specify the properties that are appropriate for that type of step.
5. Optional: Enter a description of the component step.
6. For a component step, select operations for the **CE_Operations** component in the **Component** list.
   a. Click **Add** to display the Operation Selection window.
   b. From the **Component** list, select the **CE_Operations** component.
   c. From the list of **Operations**, select one or more operations for the CE_Operations component.
   d. Click **OK** to return to the **General** tab where the selected operations are displayed.
7. In the list of operations, select each operation to display the list of parameters with their names, data types, and expressions.
8. In the **Expression** field, enter an expression for each parameter.

To create a workflow, create the routes between the steps.

Creating routes between steps:

A route determines where the work is sent when the processing of the step is complete.

To create a route between two steps, you must first open the Process Designer with the Workplace client and use it to add the steps.

For information on how to use Process Designer to validate a workflow definition, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Routes** > **Create route**.

To create a route between two steps:

1. On the workflow map, verify that an origin step and one or more destination steps exist on the map.
2. Point to the edge of the origin step until the route symbol is displayed on the cursor.
3. Click the edge of the origin step and drag to a destination step to create a route. The route is displayed as a red arrow from the origin step to the destination step.
4. In the Properties pane, enter a name for the route and specify any route conditions.
5. For multiple routes from the step, repeat the previous steps for each route.

Validate and save the workflow that you created.

Validating a workflow process:

Before you save, transfer, or start a workflow, validate the workflow definition for errors that will prevent transfer to the Process Engine. After validating the workflow definition, save your workflow.

You must use the Process Designer to create a workflow definition before you can validate it.

For information on how to use Process Designer to validate the workflow definition, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Validate, Transfer, and Launch** > **Validate using the Validate tool**.

To validate the workflow definition:

1. Click **Validate Workflow** on the Process Designer toolbar.
2. To review errors, in the Validation Results window, select an item from the **Name** list to see a description of the error.
3. Determine the file format that you want to save.

   Important: If the workflow definition document contains one workflow definition, the default file format is PEP. If the workflow definition document contains more than one workflow definition, the default file format is XPDL Version 2.0. If you saved a workflow collection in XPDL format, it cannot be saved in PEP format even if it contains only one workflow definition.
4. Save your workflow.

Tracking the progress of a workflow application

Use the Process Tracker to view a graphical representation of a workflow that is currently in progress, including the values at currently active steps and historical information retrieved from event logs. You can see the steps that were completed, when they were completed, and which steps are currently active.

If the workflow author assigned you as a tracker for a workflow, you can monitor the progress, manage participants, and modify data fields, attachments, and workflow groups that are used in the workflow. You can also complete or delete work.

For information on how to use the Process Tracker, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Tracker**.

To view the workflow in progress:
1. Open a browser and enter the following Web address: `http://<Application Engine name:portnumber>/Workplace`.
2. In the Workplace client, click the **Tasks** tab.
3. Open a Tracker item from your Tracker folder or click **Status** (if available) in a work item you opened from your Inbox.

Opening a workflow by using the Process Designer

In the Process Designer, you can open workflow definition files in two formats: the PEP format, which is the native format for Process Designer, and the XPDL 2.0 format, which is a standard format for Business Process Modeling Notation. For both formats, the workflow definition is represented as a workflow collection.

You must create a workflow definition in your local file system before you can open that workflow definition.

For information on how to use Process Designer to validate the workflow definition, in the FileNet P8 Documentation table of contents, click **User Help** > **Integrating workflow** > **Workflow applications** > **Process Designer** > **Getting started** > **Open a workflow definition**.

To open a workflow definition from your local file system:
1. Open a browser and enter the following Web address: `http://<Application Engine name:portnumber>/Workplace`.
2. In the Workplace client, click the **Author** tab.
3. On the Author page, click the **Process Designer**.
4. From the Workplace Process Designer window, click **File** > **Open**.
5. Browse to the folder where the file is stored, and select the workflow definition file. Click **Open**. This file must be of type PEP or XPDL.

Searching for documents in FileNet Business Process Manager workflow

When you design a FileNet Business Process Manager workflow application, you can search for a documents in IBM Content Manager by using the `searchForOne` or `searchForMany` methods in the `DKContentOperationsICM` class.

To search for documents in FileNet Business Process Manager workflow:
1. Open a browser and enter the following Web address: `http://<Application Engine name:portnumber>/Workplace`.
2. In the Workplace client, click the **Author** tab.
3. On the Author page, click the **Process Designer**.
4. From the Workplace Process Designer window, click **File** > **New Workflow**. A new workflow collection is created with an empty workflow definition. The workflow map displays the Launch step, which is the first step in a workflow.

5. In the Step Palette area, located below the Workflow (Main Map) area, select **BPM Palette** from the **View Palette** list. The type of steps that you want to use are displayed in the Step Palette area.
6. Drag the **Component** step from the Step Palette area to the workflow map area.
7. In the Properties pane, replace the default step name in the **Name** field with a meaningful name, for example, `MyQuery`.
8. Select operations for the **CE_Operations** component in the component list:
   a. Click **Add** to display the Operation Selection window.
   b. From the **Component** list, select the **CE_Operations** component.
   c. From the list of Operations, select either the **searchForOne** or **searchForMany** operation for use in the step.
   d. Click **OK** to return to the General tab where the selected operations are displayed.
9. Select the **searchForOne** and **searchForMany** operations to display the parameter names, parameter types, and expressions:
   a. If the parameter type is Attachment, click the **Expression** field.
   b. In the Expression builder window that opens, enter the `VWAttachment` object for `MyQuery`.

      For example, for a `VWAttachment` object for a simple query, such as `MyQuery`, enter:`"||3|4|ICMNLSDB|/NOINDEX||"`.

      For a VWAttachment object with a conditional query, enter:`"||3|4|ICMNLSDB|/NOINDEX[@USER_ID = ""Robert""]|"`
10. Validate your workflow definition.
11. Save and start your workflow.

Integration of IBM Case Manager with IBM Content Manager

The integration of IBM Case Manager with IBM Content Manager allows IBM Case Manager to work with IBM Content Manager events.

Each document and folder on an IBM Content Manager server that has been enabled for IBM Case Manager integration has a proxy document on an IBM Case Manager server that references that document or folder. In addition, each case folder on an IBM Case Manager server has a proxy case folder on an IBM Content Manager server that references that case folder.

Proxy documents and proxy folders do not contain user properties, except for an external reference to the IBM Content Manager server. All of the actual documents and folders for a case reside on the IBM Content Manager server. Document browsing is performed on the IBM Content Manager server while case browsing is performed on the IBM Case Manager server.

For each case folder on the IBM Case Manager server, there is a proxy case folder on the IBM Content Manager server that contains the case ID and the object ID of the IBM Case Manager case folder. User properties are not included in the proxy case folder. The proxy case folder item type, ICMPROXYFDRS, has the following attributes:

- ICM_CM_OBJStore
- ICM_CM_Solution
- ICM_CM_CaseType
- ICM_CM_CaseYY
- ICM_CM_CaseMM
- ICM_CM_CaseDD
- ICM_CM_CaseHH
- ICM_CM_CaseTS
- ICM_CM_CaseFdr
- ICM_CM_CaseID
- ICM_CM_CaseGUID
- ICM_CM_InitDocID

Of these attributes, only ICM_CM_CaseFdr and ICM_CM_CaseGUID are subscribed.

The case event handler is a key component of the integration of IBM Case Manager with IBM Content Manager. The case event handler is responsible for monitoring and managing the queue of case events created by the event monitor. Each case event describes an action occurring on the IBM Content Manager server for an item type that has been enabled for IBM Case Manager integration. The case event handler examines the event and, if appropriate, performs the necessary actions on the IBM Case Manager server to maintain a consistent state between the IBM Content Manager document and the IBM Case Manager proxy document, and between the IBM Case Manager case folder and the IBM Content Manager proxy case folder.

For more information about the case event handler, see the `CMBCaseEventHandler` Java class in the *Application Programming Reference*.

"IBM Content Manager events used in IBM Case Manager integration"

IBM Content Manager events used in IBM Case Manager integration

Certain IBM Content Manager events are used for IBM Case Manager integration. An IBM Content Manager event is generated when the library server combines event subscription information with a library server event.

The case event monitor converts each IBM Content Manager event to a Common Base Event (CBE) formatted event and writes the event to the event monitor database queue (ICMSTEVENTMONITORQUEUE table). The case event handler scans the event monitor database queue periodically, according to the interval that has been specified. If an event requires an action, the case event handler makes the appropriate call to the IBM Case Manager server to perform that action.

Creating a document to initiate a new case

When a new document is created, the following events are sent to the event monitor database queue by the event monitor:

- Create document. For this event, the case event handler determines whether the document is an originating document for a case type or an originating document for a task type. In either case, a case document is created on the IBM Case Manager server under the document class name that corresponds to the item type name of the IBM Content Manager document. A case document is a proxy document on the IBM Case Manager server and it is filed in a case folder.
- Add document to folder: For this event, the case event handler determines whether the document is already filed in a case folder. Otherwise, the case event handler searches for the proxy document. If the proxy document does not exist on the IBM Case Manager server, it is created and filed in the case folder.

Adding a document to a case

When an existing document is added to a case, the following event is sent to the event monitor database queue by the event monitor:

- Add item to folder. For this event, the case event handler determines whether the item is already filed in a case folder. Otherwise, the case event handler searches for the proxy document. If the proxy document is found, it is filed into the case folder. If the proxy document does not exist on the IBM Case Manager server, it is created and filed in the case folder. If the document class that corresponds to the item type of the proxy document exists on the IBM Case Manager server, then that class is used as the proxy document class. Otherwise, the default case document type, `CmAcmCM8ProxyDocument` is used.

Removing a document from a case

When a document is removed from a case, the following event is sent to the event monitor database queue by the event monitor:

- Remove item from folder. For this event, the case event handler unfiles the proxy document from the case folder.

Deleting a document

When a document is deleted, the following event is sent to the event monitor database queue by the event monitor:

- Delete an item. For this event, the case event handler unfiles the proxy document from the case folder and deletes it.

Reindexing an item

When an item is reindexed to an item type with a reindex event subscription, the following event is sent to the event monitor database queue by the event monitor:

- Reindex an item. For this event, the case event handler verifies that the target item type participates in case management. The case event handler searches for an existing proxy document in a case folder on the IBM Case Manager server for the document that was reindexed based on its persistent identification (PID) string from the original item type. If the proxy document does not exist on the IBM Case Manager server, it is created under the document class corresponding to the new item type name. If this new item type has been configured to initiate a new case, a new case instance is created and the proxy document is filed in that case folder. Otherwise, if a proxy document exists in a case folder, the case event handler makes calls to the IBM Case Manager server to update the PID value of the proxy document to match the PID value of the reindexed item. If the target item type does not participate in case management, no action is taken and the event is marked as processed in the event monitor database queue.

Reindexing from an item

When reindexing from an item type with a reindex from an item type event subscription, the following event is sent to the event monitor database queue by the event monitor:

- Reindex from an item type. For this event, the case event handler verifies that the source item type participates in case management. If so, the case event handler finds the proxy document on the IBM Case Manager server and updates its PID value to match the new PID of the reindexed item.

  "IBM Content Manager event formats for IBM Case Manager integration"

IBM Content Manager event formats for IBM Case Manager integration

IBM Case Manager preforms actions on IBM Content Manager items that can trigger IBM Content Manager events to be written to the event monitor queue by the case event handler. Each these events has its own event format and event code.

IBM Case Manager actions can trigger the following IBM Content Manager event codes:

301	Event code for creating an item.
303	Event code for deleting an item.
306	Event code for reindexing an item.
307	Event code for adding an item to a folder (auto-enabled or manual). These type of events are ignored by the case event handler for all folders that are not members of the ICMPROXYFDRS item type.
308	Event code for removing an item from a folder (auto-enabled or manual).

These type of events are ignored by the case event handler for all folders that are not members of the ICMPROXYFDRS item type.

311 Event code for reindexing an item when the reindex operation is monitored against the source item type.

The IBM Content Manager events that IBM Case Manager can trigger have the following formats:

Create item event format

The following event format is for an event that occurs when an item is created:

```
ICMEMSTART;ETYPE=ITEM;EACTION=CREATE;ECODE=301;
ITEMID=<item_id>;ITEMTYPE=<item_type_name>;
PID=<pid_string>;NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Delete item event format

The following example shows the event format for an event that occurs when an item is deleted:

```
ICMEMSTART;ETYPE=ITEM;EACTION=DELETE;ECODE=303;
ITEMID=<item_id>;ITEMTYPE=<item_type_name>;
VERSION=<version>;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
PROCESS=<process_name>;PVERSION=<process_version>;...;
ICMEMEND
```

Reindex item event format

The following event format is for an event that occurs when an item is reindexed:

```
ICMEMSTART;ETYPE=FOLDER;EACTION=REINDEX;ECODE=306;
ITEMID=<item_id>;OLDITEMTYPE=<old_item_type_name>;
OLDVERSION=<version>;
ITEMTYPE=<new_item_type_name>;
VERSION=<new_version>;
PID=<pid_string>;NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
```

```
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;
ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;
ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Add item to folder event format

The following event format is for an event that occurs when an item is added to a folder:

```
ICMEMSTART;ETYPE=FOLDER;EACTION=ADD;ECODE=307;
ITEMID=<folder_item_id>;VERSION=<folder_version>;
CHILDITEMID=<child_item_id>;
CHILDVERSION=<child_version>;
PID=<pid_string>;
CHILDPID=<child_pid_string>;
NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Remove item from folder event format

The following event format is for an event that occurs when an item is removed from a folder:

```
ICMEMSTART;ETYPE=FOLDER;EACTION=REMOVE;ECODE=308;
ITEMID=<folder_item_id>;VERSION=<folder_version>;
CHILDITEMID=<child_item_id>;
CHILDVERSION=<child_version>;
PID=<pid_string>;
CHILDPID=<child_pid_string>;
NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Reindex from item event format

The following event format is for an event that occurs when an item is reindexed and the reindex operation is monitored against the source item type:

```
ICMEMSTART;ETYPE=ITEM;EACTION=REINDEXFROM;ECODE=311;
ITEMID=<item_id>;
TOITEMTYPE=<to_item_type_name>;
TOVERSION=<to_version>;
ITEMTYPE=<item_type_name>;
VERSION=<version>;
PID=<pid_string>;
NCOMPTYPE=<number_of_comp_type>;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
COMPTYPE=<comp_type_name>;COMPNUMBER=<comp_number>;
NATTRS=<num_attrs>;ATTRNAME=<attr_name>;
ATTRLEN=<attr_length>;ATTRTYPE=<attr_type>;
ATTRVAL=<attr_value>;...;
ETIME=<time_stamp>;
NPROCESS=<number_of_process>;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
PROCESS=<process_name>;PVERSION=<process_version>;
PTYPE=<process_type>;
NMAPATTRNAME=<number_of_mapping_attr>;
MAPATTRNAME=<mapping_attr_name>;
```

```
CMATTRNAME=<cm_attr_name>;
MAPATTRNAME=<mapping_attr_name>;
CMATTRNAME=<cm_attr_name>;...;
ICMEMEND
```

Additional information:

- The PID string is empty if the item is deleted before the event monitor has generated the PID string for the item.
- The CMATTRNAME attribute can have one of the following formats:
  - <cm_attr_name>: The attribute is a root attribute.
  - <comp_type_name>.<cm_attr_name>: The attribute is a subcomponent attribute.

The following five system-defined attributes are logged by the library server if they are subscribed to by the administrator. System-defined attributes of the root component are always used:

CREATEUSERID
: Create user ID.

CREATETS
: Create timestamp.

LASTCHANGEDUSERID
: Update user ID.

LASTCHANGEDTS
: Update timestamp.

EXPIRATIONDATE
: Expiration timestamp.

Understanding prefetching in IBM 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 Content Manager for z/OS 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"

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. The following figure depicts the flow of a prefetch transaction:

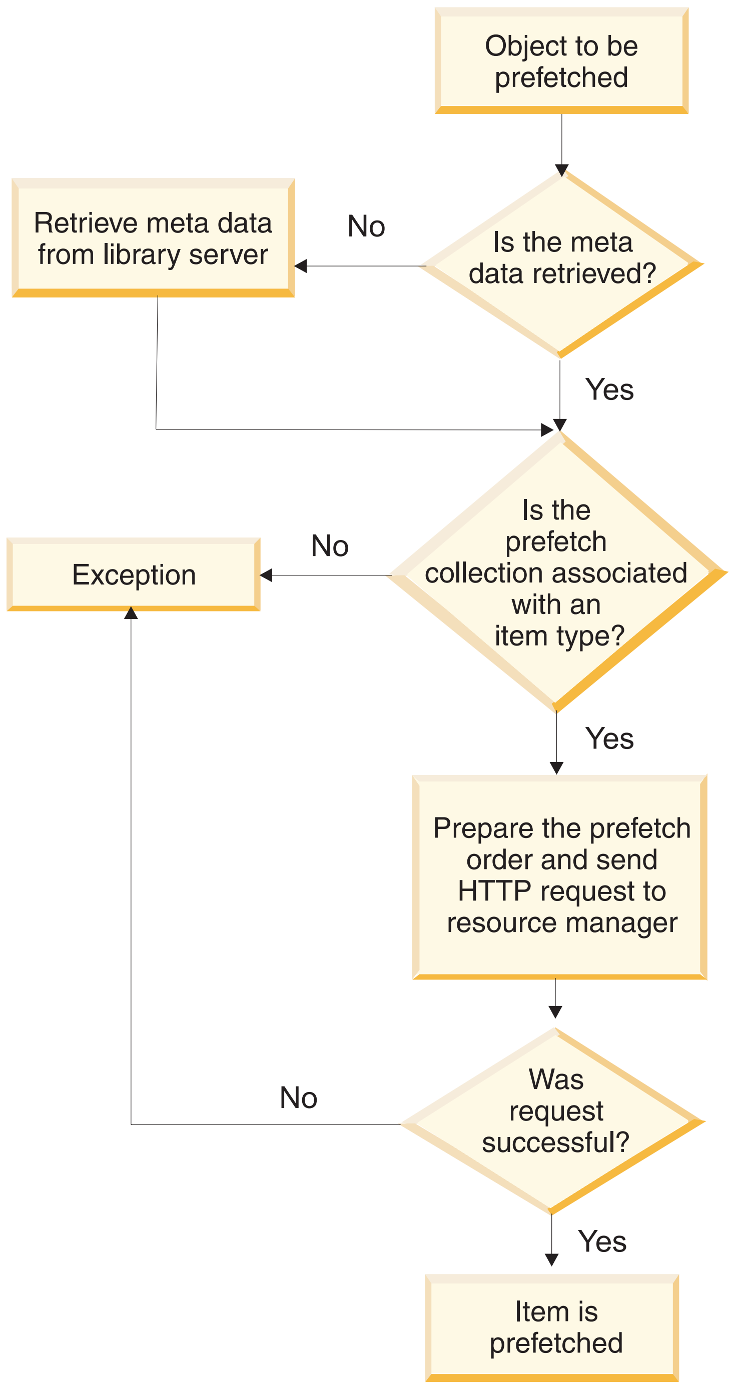

Object to be prefetched
Is the meta data retrieved?
No
Retrieve meta data from library server
Yes
Is the prefetch collection associated with an item type?
No
Exception
Yes
Prepare the prefetch order and send HTTP request to resource manager
Was request successful?
No
Yes
Item is prefetched

Figure 11. 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 as follows:

1. The application calls the resource manager by 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 exists only in its original location or source collection. The table update process is as follows:

a. An entry is inserted into the ICMRMPREFETCH table.
b. 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 ICMMOSAP asynchronous process. This process performs a `select` on the ICMRMCONTROL table for PREFETCHENABLED. If this value is set to 1, ICMMOSAP processes the entries in the ICMRMPREFETCH table if any exist. This process is as follows:
   a. PREFETCHSTATE is updated to WORKING and 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, PREFETCHSTATE is updated to COMPLETE and 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 methods is limited to Java only. The following methods are the methods 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 these methods, see the *Application Programming Reference*.

"Table definitions related to prefetching"

Table definitions related to prefetching

The ICMRMCONTROL table describes the table definitions related to prefetching.

Table 45. ICMRMCONTROL

PREFETCHENABLED	SMALLINT	NOT NULL,

Table 46. ICMRMPREFETCH

ITEMID	**CHAR(26)**	**NOT NULL**
VERSIONID	SMALLINT	
VERSIONID	SMALLINT	
EXTOBJNAME	CHAR(44)	NOT NULL
SOURCECOLLNAME	CHAR(44)	NOT NULL
PREFETCHCOLL	NAME CHAR(44)	NOT NULL
REQUESTTIMESTAMP	TIMESTAMP	NOT NULL

Table 46. ICMRMPREFETCH (continued)

ITEMID	CHAR(26)	NOT NULL
STATETIMESTAMP	TIMESTAMP	NOT NULL
OAMRETURNCODE	SMALLINT	
OAMREASONCODE	CHAR(8)	
VOLSER	CHAR(6)	
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 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 that PREFETCHSTATE was modified. For example, when an object is initially requested to be prefetched, the REQUESTTIMESTAMP and the STATETIMESTAMP are the same and PREFETCHSTATE is set to INITIATED. Once the processing of the prefetch request begins, PREFETCHSTATE is updated to WORKING. STATETIMESTAMP is also updated to reflect the time the prefetch processing began. If the prefetch processing completes successfully, PREFETCHSTATE is updated to COMPLETE. If there is a failure, PREFETCHSTATE is updated to FAILED. In either case, 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 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 following list 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 prefetch request of the object.

  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 IBM Information Integrator for Content. Each content server has a separate content server class.

To create a content server, use the `DKDatastore`*xx* classes, where *xx* identifies the specific content server. The following table shows these classes.

Table 47. Server type and class name terminology

Content server	Class name
IBM Content Manager Version 8.4	`DKDatastoreICM`
Content Manager OnDemand	`DKDatastoreOD`
IBM Content Manager for AS/400 (VisualInfo for AS/400)	`DKDatastoreV4`
Image Plus 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`. An example of a concrete class for this interface is `DKDatastoreDefICM`.

dkEntityDef
: Uses the methods to access entity information and 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. An example of a concrete class for this interface is `DKItemTypeDefICM`.

dkAttrDef
: Defines methods to access attribute information and to create and delete attributes. An example of a concrete class for this interface is `DKAttrDefICM`.

dkServerDef
: Defines methods to access server information. An example of a concrete class for this interface 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. An example of a concrete class for this interface is `DKLobICM`.

The data definition classes and their class hierarchy are represented in the following figure:

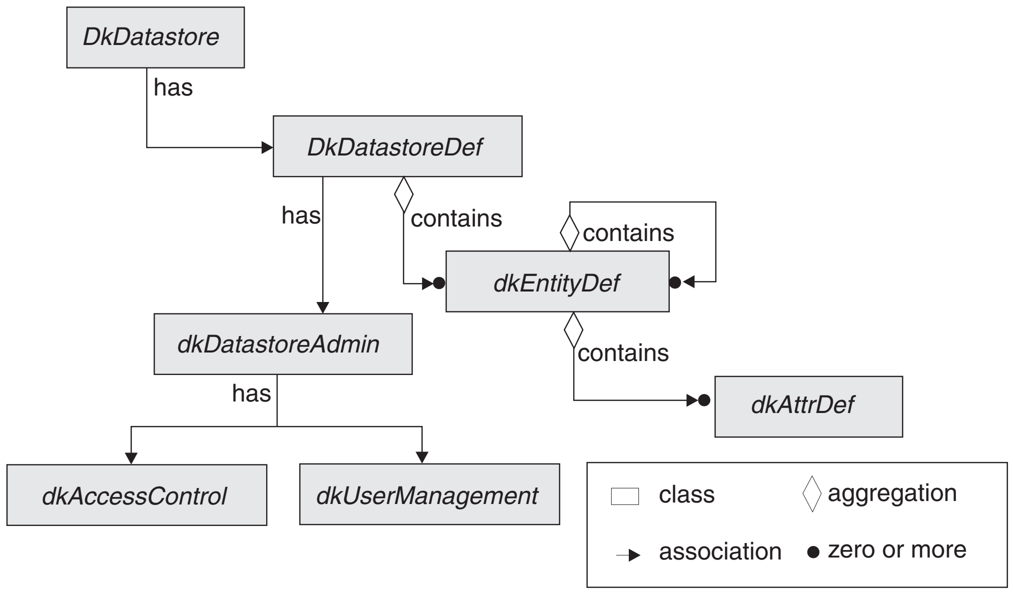


Figure 12. Data definition class hierarchy

"Working with Content Manager OnDemand"

"Working with Image Plus 390" on page 382

"Working with IBM Content Manager for AS/400" on page 389

Working with Content Manager OnDemand

IBM Information Integrator for Content provides a connector and related classes for accessing content on Content Manager OnDemand servers.

This functionality is deprecated in IBM Content Manager Version 8.4.

Important: Content Manager OnDemand does not support Net Search Extender or Combined query.

"Representing Content Manager OnDemand servers and documents" on page 371

"Connecting to and disconnecting from the Content Manager OnDemand server" on page 371

"Listing information about Content Manager OnDemand" on page 372

"Retrieving an Content Manager OnDemand document" on page 375

"Enabling the Content Manager OnDemand folder mode" on page 379

"Asynchronous search" on page 380

"Content Manager OnDemand folders as search templates" on page 380

"Content Manager OnDemand folders as native entities" on page 380

"Create and modify annotations" on page 381

"Tracing" on page 381

Representing Content Manager OnDemand servers and documents

You represent a Content Manager OnDemand content server by using a `DKDatastoreOD` in an IBM Information Integrator for Content application. You represent an OnDemand document as a DDO by using a `DKDDO`.

Content Manager 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

The attributes of a Content Manager OnDemand document are stored in a `DKDDO` as properties. The segments and notes of a Content Manager OnDemand document are stored as `DKParts`.

All other document data (resource group and views, both fixed and logical), are stored as special properties in a Content Manager OnDemand DDO with the following property names are reserved for the Content Manager OnDemand:

DKViews
A collection of logical views

DKFixedView
Contains fixed view information

DKResource
Contains resource group data

Important:

1. An IBM Information Integrator for Content administrator must properly define the Content Manager OnDemand connector by specifying the string in the **Additional Parameters** field on the Initialization Parameters page: `ENTITY_TYPE=TEMPLATES;;`. Be sure to include the two semicolons.
2. In IBM Information Integrator for Content Version 8.3, `DKViews`, `DKFixedView`, and `DKResource` replace `DKViewDataOD`, `DKFixedViewDataOD`, and `DKResouceGrpOD`.

Connecting to and disconnecting from the Content Manager OnDemand server

To connect to the Content Manager OnDemand server, use the `connect` method and to disconnect from the server, use the `disconnect` method

To log in to a Content Manager 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 Content Manager OnDemand server.

Example: Java

```
System.out.println("disconnecting from the datastore ...");
dsOD.disconnect();
dsOD.destroy(); // Finished with the datstore
```

Example: C++

```
cout << "disconnecting from the datastore ..." << endl;
dsOD->disconnect();
delete dsOD;
```

Listing information about Content Manager OnDemand

You can list application groups and folders for Content Manager OnDemand servers.

“Listing application groups”

“Listing Content Manager OnDemand folders” on page 374

Listing application groups

You can list application groups in Content Manager OnDemand by 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 Content Manager OnDemand folders

To get a list of folders in the Content Manager 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 Content Manager OnDemand document

In the Content Manager OnDemand server, you can retrieve documents. You can also display documents with their parts and attributes.

"Searching for a particular document"

"Displaying documents and their parts and attributes" on page 377

Searching for a particular document

You can search against an application group (`OnDemand Publications`) in the Content Manager 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

You can 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;

// continued...
    if (pDO.protocol() == DK_XDO)
    {
      System.out.println("     dkXDO object " + i + " in collection");
      pXDO = (dkXDO)pDO;
      DKPidXDO pid2 = pXDO.getPidObject();
      System.out.println("           XDO pid string: " +
                                       pid2.pidString());
              //----- Retrieve and open instance handler for an XDO
              pXDO.retrieve();
              // pXDO.open();
           }
         }
       }
       else if (a != null)
       {
         System.out.println("       Attribute Value: " + a.toString());
         if (strDataName.equals("DKResource")  ||
             strDataName.equals("DKFixedView") ||
             strDataName.equals("DKLargeObject"))
         {
           pDO = (dkDataObjectBase)a;

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

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 Content Manager OnDemand folder mode

To enable the Content Manager OnDemand folder mode, the string `ENTITY_TYPE=TEMPLATES` must be passed to the Content Manager OnDemand connector as part of the connection string or the configuration string.

Configuration strings are shown in the following code examples:

Example: Java

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

Example: C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

Connection strings are shown in the following code examples:

Example: Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
dsOD.connect(hostname, user ID, password, "ENTITY_TYPE=TEMPLATES");
```

Example: C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
dsOD->connect(hostname, user ID, password, "ENTITY_TYPE=TEMPLATES");
```

Asynchronous search

The Content Manager OnDemand connector supports both federated and direct asynchronous searches. You can stop an asynchronous search at any time.

An asynchronous search does not busy the main thread and can be canceled 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 Content Manager 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 user ID LIKE ',` the resulting number of documents returned to the client can consume all available memory on the machine of the client. By issuing an asynchronous search that uses the `executeWithCallback()` method, you can set the value for the maximum number of documents returned and cancel the search at any time.

If your result set is too large, you might need to increase the default Java Virtual Machine (JVM) stack size. The default stack size for each Java thread is 400 K, which allows 3920 return items before the stack overflows. Increasing the JVM stack size to 800 K 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 that use asynchronous search, see the `TRetrieveWithCallbackOD`, `TRetrieveFolderWithCallbackOD` and `TCallbackOD` sample programs.

Content Manager OnDemand folders as search templates

Use visual beans for both direct and federated searches by setting `setDsType`, `setServerName`, and `setConnectString` properties before you log in.

Three of the IBM Information Integrator for Content visual JavaBeans, `CMBSearchTemplateList`, `CMBSearchTemplateViewer`, and `CMBSearchResultsView`, use IBM Information Integrator for Content search templates. You can use these beans for federated searches by setting the `CMBConnection` `dsType` to `Fed`.

You can use these beans for direct searches on the Content Manager OnDemand servers as well. Set the following properties before you log in:

```
connection.setDsType("OD");
connection.setServerName(<odserver>);
connection.setConnectString("ENTITY_TYPE=TEMPLATES");
```

Content Manager OnDemand folders as native entities

The Content Manager OnDemand connector can also map Content Manager OnDemand folders as native entities by specifying the connect string `"ENTITY_TYPE=TEMPLATES"`.

Using Content Manager OnDemand folders as entities can be useful in federated searches, where folder definitions might be easier to work with than Content Manager OnDemand application groups, the default native entity for Content Manager OnDemand.

For federated searches, specify the server definition in IBM Information Integrator for Content administration. You can then define and map federated entities to the folders on the Content Manager OnDemand server.

Create and modify annotations

When using the Content Manager OnDemand viewer, which is launched by the `CMBDocumentViewer` bean, you can create, modify, and delete annotations for Content Manager OnDemand documents by using the `CMBDataManagement` bean and the associated `CMBAnnotation` class.

Tracing

You can trace events with the Content Manager OnDemand connector.

To enable the connector Java API trace, place the trace INI file (`cmbodtrace.ini`, for Java or `cmbodCtrace.ini`, for 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, 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 lowercase.

Verify that the path name specified in the trace file is valid and that the line containing `CMBODTRACEDIR` is not preceded with a `#` sign. The following examples are sample trace INI files:

Example: Java

```
#================================================================
# This is a java property file - not a real INI file
# ***********************************************************************#
# For windows systems, be 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.
#
# Verify that the directory names are separated by two back slash
# characters to avoid undesirable results.
#
# Verify that 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. However, 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 and exit points in JNI only. Produce the least amount
# of trace information.
#
# 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.
#
# Verify that 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. However, 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 points of all C++ methods and functions.
#
# If CMBODTRACESCOPE is missing, or set to anything else, a full trace
# is taken.
#
# To disable the trace, add a leading # character in column 1 on
# the CMBODTRACEDIR line.
#
[ODCTRACE]
# For AIX: change next line to CMBODTRACEDIR=/usr/lpp/cmb/cmgmt/ctrace
CMBODTRACEDIR=D:\Ctrace
#CMBODTRACESCOPE=ENTRY_EXIT_ONLY
```

Working with Image Plus 390

IBM Information Integrator for Content APIs support a number of features when using Image Plus 390 content servers.

IBM Information Integrator for Content APIs support the following features when working with Image Plus 390 content servers:

- Connecting and disconnecting from one or more Image Plus 390 servers.
- Retrieving categories.
- Retrieving attribute fields.
- Retrieving folders.
- Retrieving documents.

You represent an ImagePlus for OS/390 content server by using `DKDatastoreIP` in your application.

Restriction: Image Plus 390 does not support:

- Net Search Extender
- Combined query
- Workbasket and workflow

“Listing entities and attributes”

“ImagePlus for OS/390 query syntax” on page 387

Listing entities and attributes

After creating a DKDatastoreIP object and connecting to an Image Plus 390 content server, you can check its entities and attributes.

The following example lists all the entities for an Image Plus 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 the attributes associated with each entity by using the getAttr and listAttrNames functions of DKEntityDefIP:

Example: Java

```
//-- List attributes by 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 by using listAttrNames
and getAttr functions" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
  docDef = (DKEntityDefIP*)(pIter->next()->value());
  cout << " Document type   : "
 << docDef->getType() << endl;
  cout << " Document type name: "
<< docDef->getName() << endl;

  long tmpCount;
  DKString* attrNames;
// Upon return, tmpCount contains the number
// of elements in the list.
  attrNames = docDef->listAttrNames(tmpCount);
  for (int i=0; i<tmpcoun; i++)
  {
    cout << "   Attr name before lookup "
 << attrNames[i] << endl;
    attrDef = (DKAttrDefIP*)(docDef->
getAttr(attrNames[i]));
    cout << "   Attr name [" << i << "]
: " << attrDef->getName() << endl;
    cout << "   Attr id
: " << attrDef->getId() << endl;
    cout << "   Entity name
: " << attrDef->getEntityName() << endl;
    cout << "   Datastore name
: " << attrDef->datastoreName() << endl;
    cout << "   Attr type
: " << attrDef->getType() << endl;
    cout << "   Attr restrict
: " << attrDef->getStringType() << endl;
    cout << "   Attr min val
: " << attrDef->getMin() << endl;
    cout << "   Attr max val
: " << attrDef->getMax() << endl;
    cout << "   Attr display
: " << attrDef->getSize() << endl;
    cout << "   Attr precision
: " << attrDef->getPrecision() << endl;
    cout << "   Attr scale
: " << attrDef->getScale() << endl;
    cout << "   Attr update   ? "
<< attrDef->isUpdatable() << endl;
    cout << "   Attr nullable ? "
<< attrDef->isNullable() << endl;
    cout << "   Attr queryable? "
<< attrDef->isQueryable() << endl;
    cout << "" << endl;
    delete attrDef;
   }  // end for

 delete [] attrNames;

}  // end while
delete pIter;
```

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

Example: Java

```
//- List attributes by 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 by 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 query for ImagePlus for OS/390 uses the search expression parameter and option keywords.

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 use only 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 by 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 applies only 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 IBM Content Manager for AS/400

The API classes provided for IBM Content Manager for AS/400 (VisualInfo for AS/400) are similar to those provided for IBM Content Manager.

Restriction: IBM Content Manager for AS/400 does not support:
- Net Search Extender
- Combined query
- Workbasket and workflow

"Listing entities (index classes) and attributes"
"Running a query" on page 391
"Running a parametric query" on page 395

Listing entities (index classes) and attributes

You represent an IBM 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 IBM 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 (`TListCatalogV4.cpp`) is available in the `samples` directory.

Running a query

You can run a query in IBM Content Manager for AS/400 and process 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());
             pDOBase = a;

             if (pDOBase->protocol() == DK_PDDO)
             {
               pDDO = (DKDDO*)pDOBase;
               cout << "  DKDDO object " << b
<< " in " << strDataName
                    << " collection " << endl;
               k = pDDO->propertyId
(DK_CM_PROPERTY_ITEM_TYPE);

               if (k > 0)
               {
                 a = pDDO->getProperty(k);
                 val = a;
cout << "  ******************************" << endl;


 switch (val)
   {
 case DK_CM_DOCUMENT :

 cout << "  Item is a document " << endl;
      break;
 }
   case DK_CM_FOLDER :
   {
 cout << "  Item is a folder " << endl;
 break;
 }
   }
cout << "  ******************" << endl;
    }
            }
// continued...
 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

To run a parametric query, first create a query string, then run the query string, and process the results.

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, 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, TSamplePQryV4.cpp, is available in the samples directory.

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 IBM 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 and data objects, and are part of the IBM Information Integrator for Content framework.

Federated classes work across different content servers, such as Image Plus 390. However, the C++ federated connector is not currently supported on Oracle. These classes provide a set of generic functions for federated search and access across the content servers. This common model, called a *federated document model*, is illustrated in the following figure.

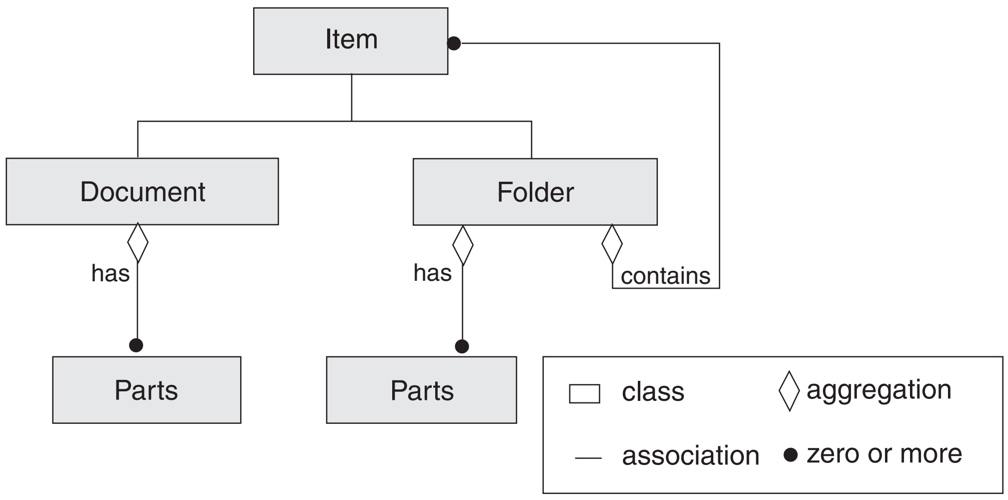


Figure 13. Federated document model

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 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). A DDO is a self-describing data object for transferring data to and from a content server. Because the DDO has a general structure that supports various 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 that is composed of attributes. An *attribute* is a label used for metadata in content servers; for example, profiles, fields, and keywords are content server attributes.

Each content server has its own terminology to explain the model it is supporting. The following table describes the terminology used for various content servers to the federated model:

Table 48. Mapping terminology for each content server

Content server	Data source	Entity	Attribute	View
IBM Content Manager Version 8.4	library server	item type	attribute	item type view or item type subset
Content Manager OnDemand	Content Manager OnDemand server	application group, folder	field, criteria	N/A
ImagePlus	ImagePlus for OS/390 server	entity	attribute	N/A
IBM Content Manager for AS/400	IBM 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"
"Using federated content server mapping components"
"Running federated queries" on page 399
"Working with system administration" on page 403

Federated schema mapping

A *schema mapping* represents a mapping between the schema in the content server and the structure of the items to process in the application.

A *federated schema* is the conceptual schema of an IBM Information Integrator for Content federated content server. It 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 IBM 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 IBM Information Integrator for Content.

The user ID and content server information are maintained in the IBM 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 by using the APIs, create a federated query string and pass it to the `execute` or `evaluate` method of the federated content server.

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

If the query is from a GUI-based application, the query does not need to be parsed and the corresponding federated query form can be constructed directly.

As a federated search is processed, IBM 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 processes them as follows:

- 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 the following figure.

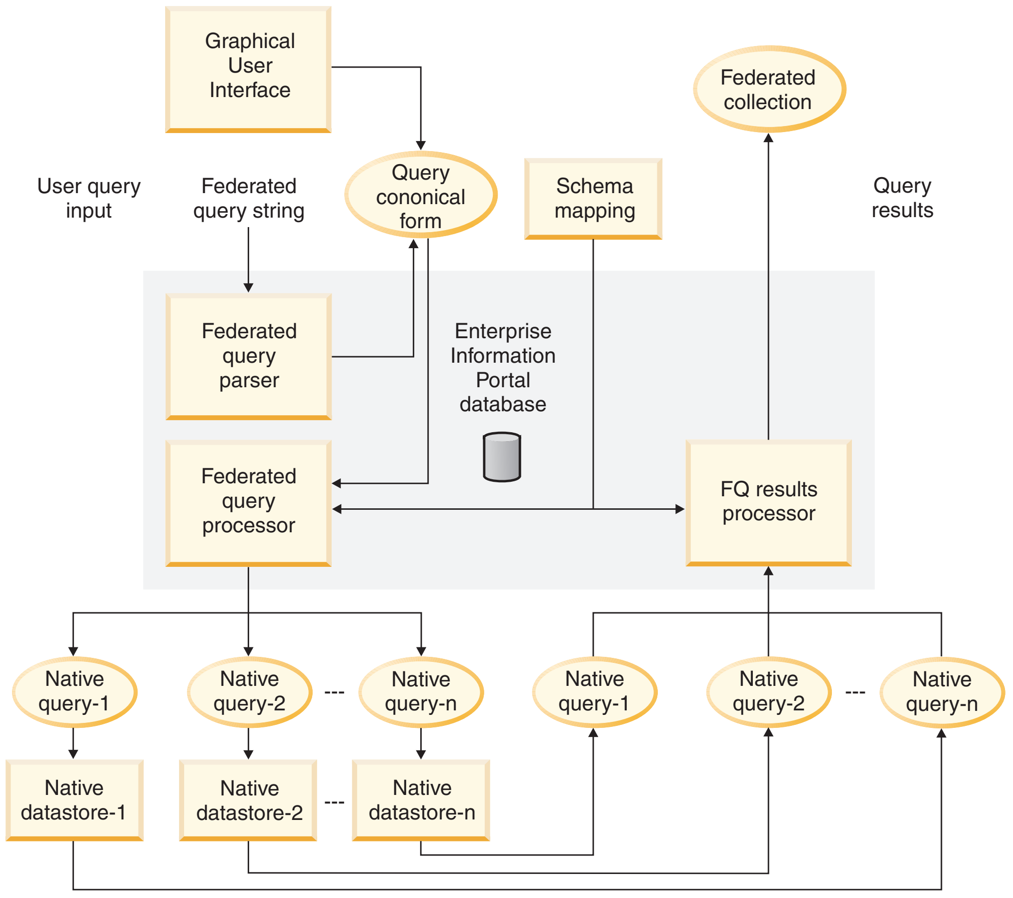


Figure 14. Federated query processing

"Federated query syntax"

"Storing query results in federated folders (Java only)" on page 402

Federated query syntax

Federated queries must have a syntax.

When you create a federated query, it must be in the correct syntax, as shown in the following example. 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 by using the LIKE operator

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

Federated parametric query by using the LIKE and > operator

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

Federated parametric query by using the LIKE and < operator

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

Federated parametric query by 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 by using the NOTBETWEEN operator

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

Federated parametric query by 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 by 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 by using the == operator

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

Federated parametric query by using the <> operator

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

Federated parametric query by 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 by using the CONTAINS_TEXT_IN_CONTENT operator

This example searches for text in the content. The text can be a word or a phrase. This example is valid only when the text-searchable federated entity

(FedTextResource) is mapped to an IBM 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 by using the CONTAINS_TEXT operator

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

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

Federated text query

This example searches for text in content. The text can be a word or a phrase. The `ASSOCIATED_ENTITY` keyword is applicable only when a federated entity is text-searchable. This example is valid only when the text-searchable federated entity (`FedEntity`) is mapped to an IBM 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 by using the OR operator

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

Federated parametric and text query by 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 by 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 by 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)

IBM Information Integrator for Content now provides special federated entities that can hold federated folders.

Federated folders can store the combined results from a federated query, such as a document from IBM Content Manager and a related document from Content Manager OnDemand. You can then send the results directly into a workflow.

IBM 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 stores only the DDO persistent identifier (PID) strings in the folders.

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

For code samples, see the `samples` directory.

Working with system administration

IBM Information Integrator for Content provides the classes for you to access system administration functions.

See the *Application Programming Reference* for information about the specific classes to use.

"Customizing the IBM Information Integrator for Content system administration client"

Customizing the IBM Information Integrator for Content system administration client

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

The IBM Information Integrator for Content system administration client supports extending the system administration application as follows:

- Replace the user and user group dialogs in the IBM Information Integrator for Content system administration client with your own dialogs.
- Add new nodes to the hierarchy in the IBM Information Integrator for Content system administration client.
- Add new menu items to the **Tools** menu in the IBM Information Integrator for Content system administration client.

You can call user exits before and after you log on to the IBM Information Integrator for Content system administration client.

Building IBM Information Integrator for Content workflow applications (deprecated)

Using the IBM Information Integrator for Content classes and APIs, you can create or extend your own applications to use the IBM 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 IBM Content Manager Version 8.4.

Connecting to workflow services

To use IBM Information Integrator for Content workflow in your applications, start by creating an instance of `DKWorkFlowServicesFed`, then connect to it. When you are finished by 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 = "icmnlsdb";
String user ID = "icmadmin";
String pw = "password";
// ----- Create a federated datastore
DKDatastoreFed dsFed = new DKDatastoreFed();
dsFed.connect(wfsrv, user ID, 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, user ID, pw,"");
```

Example: C++

```
 // ----- Create the strings for the name of the service, user ID
// -----    and Password
DKString wfsrv = "icmnlsdb";
DKString user ID = "icmadmin";
DKString pw = "password";
// ----- Create a federated datastore
DKDatastoreFed* dsFed = new DKDatastoreFed();
dsFed->connect(wfsrv, user ID, 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, user ID, 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 by using a valid workflow template, which is a workflow definition defined in the IBM 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("wfl");
// ----- Create an instance of a workflow with the workflow template name
WF.add("WD1");
// ----- Refresh the workflow object
WF.retrieve();
// ----- Construct the container object for the workflow
```

```
  DKWorkFlowContainerFed con = WF.inContainer();
  // ----- Retrieve the container data
  con.retrieve();
  // ----- Add a PID string referring to an Extended Search document
  con.setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
  con.setPriority(100);
  // ----- Update the container
  con.update();
  // ----- Start the workflow
  WF.start(con);
```

C++

```
// - Create the DKWorkFlowFed object and set the name
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF);
WF->setName("wfl");
//Create an instance of a workflow with the workflow template name
WF->add("WD1");
// ----- Refresh the workflow object
WF->retrieve();
// ----- Construct the container object for the workflow
DKWorkFlowContainerFed* con = WF.inContainer();
// ----- Retrieve the container data
con->retrieve();
// Add a PID string referring to a content item from Extended Search
con->setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
// ----- Assign a priority of 100
con->setPriority(100);
// ----- Update the container
con->update();
// ----- Start the workflow
WF->start(con);
. . .
// When you are done, clean up by deleting the container and workflow
delete con;
delete WF;
```

Terminating a workflow

You can terminate a workflow by calling the `terminate()` or `del()` function.

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 `DKWorkFlowSerivceFed` 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, thenIBM 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 by 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. 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 `DKWorkFlowSerivceFed` 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 IBM 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 IBM Information Integrator for Content workflow JavaBeans

JavaBeans provide functionality for building workflow applications. For example, you can use JavaBeans in your application to start workflow processes that follow a predefined execution path predetermined by the process manager defined on your IBM Content Manager server.

This section describes the IBM Information Integrator for Content workflow JavaBeans that you can use to connect to an IBM 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, such as 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"
"Setting up the sample data model" on page 415
"Using workflow JavaBeans in your application" on page 422

Prerequisites

You must have these components installed and configured before you can work with IBM Information Integrator for Content Workflow JavaBeans.

- IBM IBM WebSphere Application Server 4.0.4
- IBM IBM Information Integrator for Content V 8.1, Fix Pack 1 and prerequisites

  You must have installed IBM 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

  Review the 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 IBM Content Manager and IBM WebSphere Application Server

Related reference

Configuring the LDAP 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 by using the IBM Content Manager system administration client.

1. Create a privilege set called WCMPrivilegeSet and add the following privileges:
   - `AllowConnectToLogon`
   - `ItemSQLSelect`
   - `ItemTypeQuery`
   - `ItemQuery`
   - `ItemAdd`
   - `ItemSetUserAttr`
   - `ItemSetSysAttr`
   - `ItemDelete`
   - `ItemMove`
   - `ItemLinkTo`
   - `ItemLinked`
   - `ItemAddLink`
   - `ItemRemoveLink`
   - `ItemCheckInOut`
   -

- ItemAddToDomain
- ItemGetWorkList
- ItemGetWork
- ItemRoute
- ItemRouteStart
- ItemRouteEnd
- ItemUpdateWork
- ItemGetAssignedWork
- SystemDomainAdmin
- SystemDomainQuery
- SystemDefineUser
- SystemGrantUserPrivs
- SystemQueryUserPrivs
- SystemDefineGroup
- SystemQueryGroup
- SystemDefinePrivs
- SystemDefineDomain
- SystemDefineACL
- SystemDefineSemanticType
- SystemSetACL
- SystemDefineRM
- SystemDefineXdoObject
- SystemDefineSMSColl
-

`SystemSetReplicaRule`

- `SystemSetCtrlParm`
- `SystemQueryOtherDomains`
- `SystemDefineNLSLang`
- `SystemBatchCompileACL`
- `SystemDefineMimeType`
- `SystemManageKey`
- `SystemDefineAttrs`
- `SystemGetKey`
- `SystemDefineItemType`
- `SystemDefineNewKywdClass`
- `SystemQueryAllKywdClass`
- `SystemDefineLinkType`
- `SystemQueryItemType`
- `IKFAllPermissions`
- `IKFCreateCatalog`
- `IKFDeleteCatalog`
- `IKFRetrieveCatalog`
- `IKFUpdateCatalog`
- `IKFCreateCategory`
- `IKFRetrieveCategory`
- `IKFUpdateCategory`
- `IKFDeleteCategory`
-

`IKFCreateTrainingDoc`

•

`IKFRetrieveTrainingDoc`

•

`IKFUpdateTrainingDoc`

•

`IKFDeleteTrainingDoc`

•

`IKFCreateRecord`

•

`IKFRetrieveRecord`

•

`IKFUpdateRecordIKFDeleteRecord`

•

`IKFRunServerTask`

•

`IKFRunAnalysisFunc`

•

`ClientScan`

•

`ClientPrint`

•

`ClientImport`

•

`ClientExport`

•

`ClientSendMail`

•

`ClientReceiveMail`

•

`ClientReadBasePart`

•

`ClientModifyBasePart`

•

`ClientAddNewBasePart`

•

`ClientDeleteBasePart`

•

`ClientReadAnnotation`

•

`ClientModifyAnnotation`

•

`ClientReadNoteLog`

•

`ClientAddToNoteLog`

•

- `ClientModifyNoteLog`
- `ClientReadHistory`
- `ClientAdvancedSearch`
- `ClientReadFolderContents`
- `WFWorklist`
- `EIPAdminServer`
- `EIPAdminEntity`
- `EIPAdminTextEntity`
- `EIPAdminTemplate`
- `EIPAdminInfoMining`

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 49. 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 50. 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 51. 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 52. 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 53. 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 IBM Content Manager server.

Create the following users on your LDAP server and then import them into IBM Content Manager:

Table 54. Users for sample

User	Group
"Tara"	"Workflow Participants", "Content Publisher"
"Rob"	"Workflow Participants", "Project Lead"
"Greg"	"Workflow Participants", "Content Contributer"
"Dave"	"Workflow Participants", "Content Contributer"

WebSphere Application Server uses the LDAP uid attribute for authentication. By default, IBM Content Manager uses the cn user attribute. For compatibility purposes, you should change the IBM 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 decicion 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 53 on page 420).

   Optional sample setup: Add the following work nodes from the **Node** tab: **Request Change**, **Review Request**, **Make Change**, **Review Change**

Related reference

- Document routing
- Worklist
- Attributes
- Item types
- Creating users
- Creating user groups
- Creating privilege sets
- Creating access control lists
- Defining collection point

Using workflow JavaBeans in your application

Before you call the beans, you must set the datastore name in the session.

Begin by 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 IBM Information Integrator for Content workflow JavaBeans automatically login to the IBM Content Manager server when you instantiate any of the workflow beans by calling the constructor. IBM 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", "ICMNLSDB");
```

In the example above, ICMNLSDB is the datastore name.

“Example code snippets”

Example code snippets

There are common tasks that you can complete by using the IBM 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 ICMNLSDB
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 ICMNLSDB11 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 IBM Information Integrator for Content to build Java-based or Web client applications.

The IBM 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 by using the non-visual beans (as a data model), you must use them with the non-visual beans when you build 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 (thereafter referred to as beans) are reusable software components that are written in the Java programming language and can be manipulated by using beans-aware builder tools. 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 interact with beans visually, by using a builder, or manually, by calling the beans methods from a program.

Beans define a design-time interface that allows application 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 need to use special introspection and construction tools when working with beans. The pattern signatures are 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 to 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 are exposed to application builder tools.

Properties
Properties control the appearance and behavior of a bean. Builder tools perform introspection on a bean to discover its properties and to expose those properties for manipulation, which allows you to change the property of a bean 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, when an application customizes a bean in an application builder, the state is saved so that any 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, verify that the builder supports the Java Development Kit (JDK) version required by the beans.

To add the jars specified in the following section, follow the instructions for adding new jar files that are provided by the builder. Then, to add the IBM Information Integrator for Content beans, which are in `cmb81.jar`, follow the instructions for adding beans from a jar.

Note: To use the beans, you must have JDK version 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 by using the visual and non-visual beans.

"Using IBM Rational Application Developer for WebSphere Software"

Using IBM Rational Application Developer for WebSphere Software

Use non-visual beans to build applications in WebSphere Studio Application Developer.

To build servlets and JSP pages by using non-visual beans in IBM Rational® Application Developer for WebSphere Software, 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 by using the IBM Information Integrator for Content servlets and JSP taglib, you must also specify the following files:

   `%IBMCMROOT%\lib\cmbservlet81.jar`

   `%IBMCMROOT%\lib\cmbtag81.jar`

   For the tag library, you also must import the `taglib.tld` file for the IBM Information Integrator for Content 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. Because the JARs previously listed contain J2EE classes, you must include the J2EE JAR, located in: `\Program Files\IBM\Application Developer\plugins\com.ibm.etools.websphere.runtime\lib\j2ee.jar`

Invoking the IBM Information Integrator for Content JavaBeans

You can invoke IBM Information Integrator for Content JavaBeans either by calling them directly or by wiring instances of beans to other beans.

The beans in the IBM 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 IBM Information Integrator for Content beans by using request and reply events. When by using this method, remember the following:

- The `CMBConnection` 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

IBM 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 by using the IBM Information Integrator for Content and Java connector classes. You can use the beans to build Servlets or JavaServer Pages (JSP). The beans can also be used in a command line or in Windows applications.

Using the non-visual beans provides the following benefits:
- Provides a federated access mechanism and common programming model for the many different connectors that ship with IBM Information Integrator for Content.
- Allows you to program at a higher level of abstraction.
- Hides the complexity and details of individual connectors.
- Allows you to take advantage of the beans support that is built into most commercial development environments.
- Makes it easier for your application to use multiple connectors, or to migrate, without having to make major changes to your application.

Although using beans makes building basic applications easier, there are some limitations to consider. Beans to not provide the following functionality:
- Administrative and configuration functionality.
- Batch import: The beans support only single item import capabilities. If you need batch processes for importing and exporting large amounts of data, use the connector interfaces.
- Complete server-specific functionality for all connectors. Although the beans include some server-specific functionality for certain servers, they do not surface the full functionality of each server available from the Java APIs.

These limitations can, in some cases, be bridged by using accessor methods that allow access to the underlying Java APIs.

Important: The IBM 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 an EJB as the underlying connection mechanism to unstructured data repositories.

"Non-visual bean configurations" on page 429
"Understanding the non-visual beans features" on page 429

“Non-visual beans categories” on page 430
“Considerations when using the non-visual beans” on page 434
“The CMBConnection bean” on page 436
“The CMBConnectionPool bean” on page 437
“The CMBQueryService bean” on page 441
“Display names” on page 446
“Tracing and logging in the beans” on page 446
“Understanding properties and events for non-visual beans” on page 447
“Building an application by using non-visual beans” on page 447

Non-visual bean configurations

Non-visual beans have local, remote, and dynamic configurations.

The non-visual bean configuration 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 by 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 IBM 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 by using the Model View Controller (MVC) design pattern.

Note: The view component is typically composed of JSP and the controller component of servlets (such as the ones in the EJB servlet kit).

IBM Information Integrator for Content non-visual beans have the following features:

- Provide access to the schema definitions in the library server.
- Provide create, retrieve, update, delete methods for documents, simple (non-resource) items, and resource items in all the repositories supported by IBM Information Integrator for Content.
- Provide functionality to search and retrieve documents, simple items, and resource items in all the repositories supported by IBM 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 IBM Information Integrator for Content.
- Integrate and expose the functionality provided in the IBM 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. For example, 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 other.

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. Datastore beans include the following beans:

CMBConnection
This bean maintains the connection to a back-end server, which can 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 IBM Information Integrator for Content advanced workflow processes.

CMBDocRoutingDataManagement and CMBDocRoutingQueryService
Used to work with IBM Content Manager Version 8 document routing processes.

CMBDocumentServices
Used to provide document conversion, document manipulation, and annotation services.

Helper beans
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. Helper beans include the following beans:

CMBEntity

Represents data item definitions available in content management repositories. For example, for an IBM Content Manager repository, a `CMBEntity` represents both item types and child component definitions, while for a Content Manager 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`.

CMBObject
: Represents instances of resource items, base parts, and note log parts. `CMBObject` is also used to represent BLOB attributes. `CMBObject` is a helper class for `CMBDataManagement`.

CMBAnnotation
: Represents instances of annotation parts for IBM Content Manager Version 8 repositories and notes for Content Manager OnDemand repositories.

CMBPrivilege
: Provides the functionality required to retrieve privilege-related information from an IBM Content Manager or IBM Information Integrator for Content supported repository.

CMBResultData
: Represents the result of a search. Depending on the query method and options used for the query, the search result might contain a collection that holds the entire result set, a cursor to iterate over the result set, or a portion of the result set. When not using a cursor query, the `CMBResultData` object is passed directly to the `CMBSearchResults` bean. `CMBResultData` is a helper class for `CMBQueryService` and `CMBSearchResults`.

CMBResultSetCursor
: Provides a cursor to navigate the results without holding the entire result set in memory at once. `CMBResultSetCursor` is a helper class for `CMBQueryService`.

Ancillary beans

The ancillary beans are not essential in applications, but can be useful for enhancing functionality. Ancillary beans include the following beans:

CMBConnectionPool
: Used to pool `CMBConnection` beans. Because `CMBConnection` beans wrap `DKDatastore` instances, the `DKDatastore` instances can be expensive to create. Using the `CMBConnectionPool` allows reuse of the `CMBConnection` 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
XML beans take messages in the form of XML requests and replies to perform various operations on IBM Content Manager, including searching, creating, updating, batching, exporting items into XML format, exporting item type definitions as XML schemas, and performing document routing operations. XML beans use the XML support in the API for item import, item export, and schema export operations. XML beans include the following beans:

CMBXMLServices
This bean is the underlying component of IBM 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 might 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 IBM 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 IBM 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 about workflows in the system.
- Information about the work items moving through the system as part of active workflows.
- Work list-related information.
- Information about all the registered work notifications.

The following beans are the main components of the document routing support.

CMBDocRoutingManagementICM
Use this bean to create and manage document routing processes. You can obtain an instance of this bean from an instance of the `CMBConnection` object. This bean provides support for the following features:

- Starting, terminating, suspending, and resuming a document routing instance.
- Checking out a CM item contained inside a work package.
- Setting properties for work packages being routed by a document routing instance.

CMBDocRoutingQueryServiceICM
This bean provides an interface for querying information related to document routing processes. You can obtain an instance of this bean from an instance of the `CMBConnection` object. This bean provides support for retrieving the following information:

- Information relating to the work packages being routed through the document routing system.
- Information relating to the processes that are currently active in the system.
- Information about 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 by 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 the characteristics of a document.

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 subclasses of this bean. Also, the `CMBPageAnnotation` itself contains properties like the page the annotation is on and the annotation type. The subclasses include the following classes:

- `CMBArrowAnnotation`
- `CMBCircleAnnotation`
- `CMBHighlightAnnotation`
- `CMBLineAnnotation`
- `CMBNoteAnnotation`
- `CMBPenAnnotation`
- `CMBRectAnnotation`
- `CMBStampAnnotation`
- `CMBTextAnnotation`

Other beans classes
The bean classes listed in this section provide various functionality:

BeanInfo Classes
Allow for explicit exposure of the features of the IBM 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 subclasses of `CMBException` indicates a specific error condition. In each case, properties of the exception object can be used to obtain detailed error information about 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 must 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 IBM 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
Session listeners are listener classes that exist at the session-wide level. In the IBM 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 IBM 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 IBM Information Integrator for Content beans. When session-wide beans such as CMBSchemaManagement and CMBDataManagement are obtained in this way, they are already wired to the CMBConnection bean from which they are obtained. Being wired to the CMBConnection bean allows these session-wide beans 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 be used only on a single thread at any point in time. This restriction extends to all other beans that are associated with 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 that use the CMBConnectionPool bean. Therefore, the recommendation is for each thread to 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. Therefore, an instance of the session-wide beans, such as CMBSchemaManagement, can be used only by a single thread at any given time. If the session-wide bean instance is used by multiple threads, 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 listen to connection-reply events generated by the CMBConnection beans to allow the beans to recognize whether the underlying content repository with which the CMBConnection bean instance is associated has been connected, thus ensuring that they can take the appropriate action.

Important: The connection reply events are only sent when the data store is connected or disconnected as a result of a CMBConnection bean method call.

Unlike the CMBConnection bean, 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. It is recommended that you return any connection obtained from the connection pool as soon as possible after you use it. Doing so ensures that it is available to other threads that might be requesting connections from the pool.

Memory considerations

An application that handles large numbers of items (for example, a query that can return a large result set) must perform special actions when using JavaBeans to ensure that the available memory on the system is not exceeded. There are several memory management techniques in the JavaBeans that can help reduce the amount of memory that is used by an application.

The `runQueryWithCursor` method has a significant advantage over the `runQuery` method when working with large result sets. The `runQueryWithCursor` method returns a `CMBResultData` object that contains a `CMBResultSetCursor`. The `CMBResultSetCursor` is recommended for handling large result sets in the JavaBeans. Because the cursor does not maintain references to the result items, avoid holding the entire result set in memory by using the cursor navigation methods to get each result individually and then releasing the memory from each result item as soon as you have processed it. By contrast, the `runQuery` method returns a `CMBResultData` object that contains a collection holding the entire result set. Because the collection maintains a reference to each item in the result set as you process the results, your application holds all the items in the result set in memory when it reaches the end of the result set.

To prevent a memory shortage, avoid placing all the items from a large result set in memory at the same time. The `CMBObjectManagement` Java bean is designed to help you maximize reusing object instances. The `CMBSearchResults` and `CMBResultSetCursor` beans use the `CMBObjectManagement` bean to create all `CMBItems` in the search result set. After you have finished working with a `CMBItem` or `CMBObject`, return it to the `CMBObjectManagement` bean by calling the `releaseObject` or `releaseItem` methods. After you have released the `CMBItem` or `CMBObject`, it will be cleared of all its data and made available for reuse by your application. Another way to reduce the memory used by the application is to free the resource content from an item when you no longer need the data. Call the `CMBObject.getXDO().setNull()` method to release all references to the memory that was used to store the resource content, making the memory available for garbage collection.

All these memory management techniques are demonstrated in the JavaBeans sample, `TCursorSearch.java`, located in `db2cmv8/samples/java/beans`.

The CMBConnection bean

The `CMBConnection` bean provides connection management functions for all the content server types supported by IBM Information Integrator for Content.

The `CMBConnection` bean wraps an instance of the `DKDatastore` object that is used to connect to the server. The `CMBConnection` bean is required to use other Java beans. Most of the Java beans have a connection property that is set to a single instance of the `CMBConnection` bean for the duration of the session. The typical use of the `CMBConnection` bean is to construct an instance, set properties, and then perform a connect operation. For example:

```
CMBConnection connection = new CMBConnection();
connection.setDsType("ICM");
connection.setServerName("icmnlsdb");
connection.setUserid("icmuser1");
connection.setPassword("password");
connection.connect();
```

When the session is completed, the connection is disconnected.

```
connection.disconnect();
```

Important: When a CMBConnection object is no longer referenced, it might be garbage-collected. When it is garbage-collected, the finalizer is called, and the underlying DKDatastore object is disconnected. Therefore, if your application is using a DKDatastore instance that was obtained by calling the getDs() method on a CMBConnection object, you must maintain a reference to the CMBConnection object. Otherwise, your datastore instance might be disconnected.

Connection and configuration parameters.

The CMBConnection bean allows you to set two strings that are passed to the underlying datastore to set the server-specific options in the connection. Because each server defines its own format for the connect string and configuration string parameters, you must check the documentation for the specific server type to understand the supported options.

Connect string
: The connect string is sent as a parameter to the connect method of the server. You can set the connect string by calling the setConnectString method before connecting the CMBConnection object.

Configuration string
: The configuration string is passed as a parameter to the constructor of the datastore. You can set the configuration string by calling the setConfigString method on the CMBConnection object. You must set the configuration string before calling any other methods on the CMBConnection object. If you call any other methods, a datastore might be constructed even though the method that you called did not create a connection. For example, if you set the server type to ICM, and then call the listDataSources() method to get a list of available IBM Content Manager servers, a DKDatastoreICM instance is created for the CMBConnection bean. If you then set the configuration string and call connect, you cannot pass the configuration string to the constructor because the ICM datastore has already been constructed.

Authentication options when connecting

When connecting by using the CMBConnection bean, you set a user ID and password to authenticate a user to the server. When connecting to the federated servers or IBM Content Manager servers, you also have the option of connecting by using a credential. When using a credential to connect to the server, you call the connectWithCredential method instead of the connect method and pass in a WebSphere credential object as a parameter.

Threading considerations

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. This restriction means that separate connections must be created for each thread.

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 returns 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 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 the credentials of another user.

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 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 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 the credentials of the connection being freed is activated, and the connection that was just freed is 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 returns 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 use of your application 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. Although this situation does not necessarily represent an error in the pool, it might indicate a situation in which 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 `DKDatastore` objects 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 the specific needs of their application. The optimal settings for these parameters varies 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 reused 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 and create connections to fulfill a `getConnection` request.

The following properties of the connection pool are used to control the overall memory footprint of the pool 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, which 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 `CMBConnectionPool.CMB_NO_MAX`. The default value is 100. Set the value of `maxConnections` property to at least to 2 to 3 times the number of different user IDs that access 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 results 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 `CMBConnectionPool.CMB_NO_MAX`. This value 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 various 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. Setting the maxFreeConnections property to a slightly smaller value than the maxConnections ensures that there is always a little room left in the pool for other users when they request a connection. For example, you can set maxFreeConnections according to this formula:

```
maxConnections = number of different user IDs * MaxConnectionsPerUserid
maxFreeConnections = maxConnections - number of different 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. Do not 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 CMBConnectionPool 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 many 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 waits 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 returns 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 IBM Content Manager connections and you have WebSphere Application Server connection pooling enabled for the datastore, do 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 prevents the underlying database connection from timing out.

The CMBQueryService bean

The `CMBQueryService` bean provides search functions to all the server types supported by IBM Information Integrator for Content. You can perform synchronous, asynchronous, or cursored searches.

There are two ways in which a query can be provided to CMBQueryService:

Template-based queries
Takes a `CMBSearchTemplate` instance in which the criteria has been supplied with operators and values and performs a search. For servers that support search templates, which includes Federated and OnDemand folders, a template-based search uses an administrator defined set of search criteria with allowed operators and values.

Query string
Takes a query string and an optional set of query parameters. The syntax of the query string and possible parameters are documented in the Javadoc for the server type that you are connecting to. By using this style of search, any query that is supported by the server can be performed. However, the query string syntax is different based on the server type. You can provide the query string yourself or use the `CMBQueryService.generateQueryExpression` method to generate a simple query string by providing the name of an entity and the names, operators, and values of attributes. This style of query generation supports the building of basic queries. It is not possible to use the advanced capabilities of the query languages supported by native servers when using this method to create a query.

Query parameters

Some servers allow you to specify the query parameters for the query. Any query parameter that can be set on a query to the server can be set on the `CMBQueryService` bean by using the inner class, `QueryParameter`. See the Javadoc for the specific server for definition of the allowed name/value pairs.

Sample

The following code example shows you can specify a query parameter in IBM Content Manager queries:

```
DKRetrieveOptionsICM dkRetrieveOptions =
 DKRetrieveOptionsICM.createInstance(dsICM);
dkRetrieveOptions.baseAttributes(true);
DKNVPair[] queryParams = new DKNVPair[2];
queryParams[0] = new DKNVPair(DKConstant.
 DK_CM_PARM_RETRIEVE, dkRetrieveOptions);
queryParams[1] = new DKNVPair(DKConstant.
 DK_CM_PARM_END, null);
dsICM.evaluate("/NOINDEX", DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
```

The following code example shows you can set the same query in the beans:

```
dkRetrieveOptions = DKRetrieveOptionsICM.
createInstance((DKDatastoreICM)queryService.
getConnection().getDatastore());
dkRetrieveOptions.baseAttributes(true);
CMBQueryService.QueryParameter[] queryParams =
new CMBQueryService.QueryParameter[1];
queryParams[0] = queryService.new QueryParameter
(DKConstant.DK_CM_PARM_RETRIEVE, dkRetrieveOptions);
queryService.setQueryParameters(queryParams);
queryService.setQueryString("/NOINDEX");
queryService.runQuery();
```

"Synchronous, asynchronous, and cursored search"

"Callback functions for queries that time out or are cancelled" on page 445

Synchronous, asynchronous, and cursored search

You can run a search by using any of the following methods: synchronous, asynchronous, or cursored search. You can start a search through method calls or by sending an event.

There are three types of search:

Synchronous search
: Sends the query to the server and obtains all results before returning. This type of search is the simplest form of search, but it might require a long time to run depending on the number of results obtained. It is recommended for small result sets or in cases in which an application requires the entire result set to be retrieved before displaying it.

Asynchronous search
: Sends the query to the server and then immediately returns. A secondary thread is created internally, and callbacks are made to a callback class provided to indicate that results have arrived. The callback class is called with portions of the result set until all query results have been returned. Using this type of search typically allows faster first-page display for queries.

Cursored search
: Sends the query to the server and obtains a cursor that can be used to retrieve the results from the server when required. This type of search is like an asynchronous search but it requires the application to pull the results from the server, rather than having them pushed by the server by using a callback. Additional resources are required on the server for maintaining the search results for this type of search. The cursored search is recommended for searches that are expected to have large result sets. The reason for this is that only a cursored query allows you the option of iterating through the result set without having the entire result set loaded into memory at the same time. Synchronous and asynchronous searches both use collections that maintain a reference to the item until you clear the query results. Only the cursor allows you to process the entire result set without maintaining references to all the items. The Java beans sample `TCursorSearch` demonstrates the best usage strategies to minimize the amount of memory used when processing the results.

Starting a search

To initiate a query string search using methods, set the properties on the bean for synchronous or asynchronous search, maximum search results, and the version. For simple searches, you can call `generateQueryExpression` method to obtain a query string for searching an entity by specifying a set of attribute names, operators, and attribute values. Otherwise, you must generate the query string yourself.

You can use the following methods for different search operations:

setQueryString
: Queries string-based search.

runQuery
: Starts a search.

getResults
: Obtains the results of synchronous searches. The `getResults` method returns a `CMBResultData` object. For asynchronous searches, the `CMBSearchReplyEvents` are also generated when the results are obtained from the server. The frequency of these events is set using the `setCallbackThreshold` method.

CMBResultData.setResults
: Provides the results to the `CMBSearchResults` bean.

CMBResultData.getResultObject()
: Obtains the `CMBResultSetCursor` from a cursor search. The `CMBResultData.getResultObject()` method processes the query results by iterating through the cursor. Use the `CMBSearchResults` bean to process the results for non-cursored searches and the `CMBSearchResults.setResults` method to provide the results to the `CMBSearchResults` bean.

To cancel an asynchronous query, you can call the `CMBQueryService.cancelQuery()` method. For Content Manager V8 datastores, calling the `CMBQueryService.cancelQuery()` method stops the query operation on the server and frees the resources that were used to run the query.

You can also set a timeout on the `CMBQueryService` bean that defines the maximum amount of time you want a query to run on the server. If the query takes longer than the defined timeout value, the search operation is stopped and an error is returned. For synchronous searches, a `CMBException` message is thrown, with an error code indicating that the query timed out.

Synchronous search example

The following code segment illustrates setting a synchronous query string search and iterating through the results:

```
CMBQueryService queryService = connection.getQueryService();
queryService.setAsynchSearch(false);// synchronous search
queryService.setMaxResults(1000);// limit the result set
queryService.setVersion(CMBBaseConstant.CMB_LATEST_VERSION);
 // only return the most recent version of an item
CMBQueryService.QueryCriterion crit1 =
queryService.new QueryCriterion("Source", "=",
  new String[] {"Import"});
String queryString = queryService.generateQueryExpression
("NOINDEX", new CMBQueryService.QueryCriterion[] {crit1},
   CMBBaseConstant.CMB_QS_TYPE_XPATH, true);
queryService.setQueryString(queryString,
CMBBaseConstant.CMB_QS_TYPE_XPATH);
queryService.runQuery();
CMBResultData results = (CMBResultData)queryService.getResults();
// results are always a CMBResultData object
CMBSearchResults searchResults = new CMBSearchResults();
searchResults.setConnection(connection);
searchResults.newResults(results);
for (int i = 0; i < searchResults.getCount(); i++)
{
    CMBItem item = searchResults.getItem(i);
}
```

Asynchronous search example

To run this query using asynchronous search, you would essentially use the same process. The only difference is that you would receive the results in a callback class rather than getting them from the `CMBQueryService` bean.

See the following example of how to run this query asynchronously:

```
// First define the search callback class
   class SearchReplyListener implements CMBSearchReplyListener {
      CMBQueryService queryService;
      CMBSearchResults searchResults;

      public SearchReplyListener(CMBQueryService queryService,
        CMBSearchResults searchResults) {
        this.queryService = queryService;
        this.searchResults = searchResults;
      }

    public void onCMBSearchReply(CMBSearchReplyEvent evt) {
      try {
        if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_RESULT_NEW) {
          // Always use newResults for the first set of results
          searchResults.newResults(queryService.getResults());
        } else if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_RESULT_MORE) {
         searchResults.appendResults(queryService.getResults());
        } else if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_RESULT_END) {
          searchResults.appendResults(queryService.getResults());
        }
        else if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_FAILED) {
         Throwable t = (Throwable)evt.getData();
         if (t != null)
           t.printStackTrace();
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
```

Run the query by using the following code example:

```
CMBQueryService queryService = connection.getQueryService();
CMBSearchResults searchResults = new CMBSearchResults();
searchResults.setConnection(connection);
queryService.setAsynchSearch(true);  // asynchronous search
queryService.setMaxResults(1000); // limit the result set
queryService.setVersion(CMBBaseConstant.CMB_LATEST_VERSION);
 // only return the most recent version of an item
CMBQueryService.QueryCriterion crit1 =
queryService.new QueryCriterion("Source", "=", new String[]
{"Import"});
String queryString = queryService.generateQueryExpression
("NOINDEX", new CMBQueryService.QueryCriterion[] {crit1},
      CMBBaseConstant.CMB_QS_TYPE_XPATH, true);
queryService.setQueryString(queryString, CMBBaseConstant.CMB_QS_TYPE_XPATH);
queryService.addCMBSearchReplyListener
(new SearchReplyListener(queryService, searchResults));
queryService.runQuery();
```

Cursored search example

The AsynchSearch property is not applicable to cursored queries. To run the same query as the previous example by using a cursor, perform the following steps:

```
CMBQueryService queryService = connection.getQueryService();
 queryService.setMaxResults(1000); // limit the result set
 // only return the most recent version of an item
 queryService.setVersion(CMBBaseConstant.CMB_LATEST_VERSION);
 CMBQueryService.QueryCriterion crit1 = queryService.new QueryCriterion
   ("Source", "=", new String[] {"Import"});
 String queryString = queryService.generateQueryExpression("NOINDEX",
   new CMBQueryService.QueryCriterion[] {crit1},
   CMBBaseConstant.CMB_QS_TYPE_XPATH, true);
 queryService.setQueryString(queryString, CMBBaseConstant.CMB_QS_TYPE_XPATH);
 queryService.runQueryWithCursor();
 // results are always a CMBResultData object
 CMBResultData results = (CMBResultData)queryService.getResults();
 // To get the memory benefits of using a cursor, you must use the
 // CMBResultSetCursor to iterate through the results
 CMBResultSetCursor cursor = (CMBResultSetCursor)data.getResultObject();
 while (cursor.isValid() && !cursor.isEnd()) {
   CMBItem result = cursor.fetchNext();
 }
```

Callback functions for queries that time out or are cancelled

For an application using an asynchronous search, the callback function needs to be able to distinguish between a query that has timed out, a query that has been cancelled, and a query that has ended for some other reason. Checking the eventId property of the CMBSearchReplyEvent helps you to find out why the query stopped.

Example of a query callback listener class

This example shows a query callback listener class that can distinguish between a timed out query, a cancelled query, and a failed query:

```
  class SearchReplyListener implements CMBSearchReplyListener {
      CMBQueryService queryService;
      CMBSearchResults searchResults;
      public SearchReplyListener(CMBQueryService queryService,
   CMBSearchResults searchResults)
   {
        this.queryService = queryService;
        this.searchResults = searchResults;
      }
      public void onCMBSearchReply(CMBSearchReplyEvent evt)
   {
   if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_RESULT_NEW)
   {
     searchResults.newResults(queryService.getResults());
   // Always use newResults for the first set of results
   }
   else if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_RESULT_MORE)
   {
      searchResults.appendResults(queryService.getResults());
   }
   else if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_RESULT_END)
   {
      searchResults.appendResults(queryService.getResults());
   }
   else if (evt.getStatus() == CMBBaseConstant.CMB_STATUS_FAILED)
   {
      if (evt.getID() == CMBSearchReplyEvent.CMB_REPLY_CANCEL_SEARCH)
    {
        searchResults.appendResults(queryService.getResults());
        System.out.printnl("Query cancelled");
      }
      else if (evt.getID() == CMBSearchReplyEvent.CMB_REPLY_SEARCH_TIMEOUT)
    {
        searchResults.appendResults(queryService.getResults());
        System.out.println("Query timed out");
      }
      else {
             Throwable t = (Throwable)evt.getData();
             if (t != null)
               t.printStackTrace();
          }
        }
      }
    }
```

Display names

In IBM 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 is the non-display name).

To specify which language display names that you want to appear in your application, call `CMBConnection.setLanaguageCode(String languageCode)`. The display names that are returned by the beans are translated into the language that you specify in the `languageCode` property.

Both `CMBEntity` and `CMBAttribute` contain three methods to retrieve the name and display name:

getNonDisplayName()
: Returns the real, non-display name.

getDisplayName()
: Returns the display name translated into the language specified by the `languageCode` property of the `CMBConnection` bean.

getName()
: If the `displayNamesEnabled` property is set to false (default value), `getName()` returns the non-display name. Otherwise, if the `displayNamesEnabled` property is set to true, `getName()` returns the display name. The display name is translated into the language specified by the `languageCode` property of the `CMBConnection` bean.

  Important: The `displayNamesEnabled` property is deprecated; do not set it. If the `displayNamesEnabled` property is not set, the `getName()` method returns the non-display name.

Tracing and logging in the beans

You can enable tracing on all session-wide beans in the IBM Information Integrator for Content beans layer.

Enabling tracing on an instance of the `CMBConnection` bean also enables tracing on any IBM Information Integrator for Content bean obtained from this connection bean, including schema management, data management, query service, and workflow beans. For the tracing 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 might have a constrained property whose value is read by another bean. 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 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` automatically fires `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`. When it is added as a listener to an instance of the `CMBQueryServiceBean`, which is the source of the `CMBSearchReplyEvents`, it automatically processes the `searchReplyEvents` fired by that bean.

Building an application by using non-visual beans

You can use non-visual beans to build an application.

"A sample non-Graphical User Interface (GUI) application"

A sample non-Graphical User Interface (GUI) application

You can use 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 IBM 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 by by 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 IBM Information Integrator for Content or other content servers into Java applications based on Swing.

Visual beans perform basic tasks that are common to many applications, such as logging on, searching, displaying and viewing results, updating documents, and viewing version information.

Each visual bean has a `Connection` property. This property must reference an instance of `CMBConnection`, the nonvisual bean that maintains the connection to the content servers. Any application built with the IBM Information Integrator for Content visual beans must also contain an instance of the `CMBConnection` nonvisual bean.

CMBLogonPanel bean

This bean displays a window to log in to Content Manager EE or to content servers such asIBM Information Integrator for Content 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 displays a window that lets users log in to a content server, update user mappings, and change a password.

In the `CMBLogonPanel` bean, when a user clicks **Change**, the Change Password window appears. The user enters the old password, and enters the new password twice.

In the `CMBLogonPanel` bean, when a user clicks **Update Mapping** in the Logon window, the Update Userid Mapping window is displayed. 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 IBM Information Integrator for Content federated database.

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 (optional) 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 the window without saving 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 15, uses one image for the backgrounds of the selected items and another for the unselected items. Figure 16 on page 450 shows the simple template list style. Figure 17 on page 450 shows the drop-down template list style.

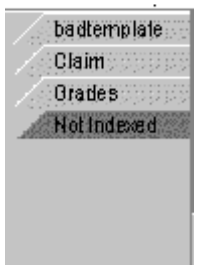


Figure 15. Image template list style

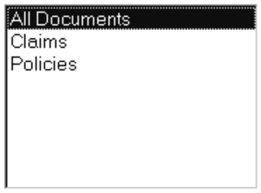


Figure 16. Simple template list style

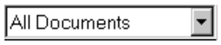


Figure 17. Drop-down template list style

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 the following figure) 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.

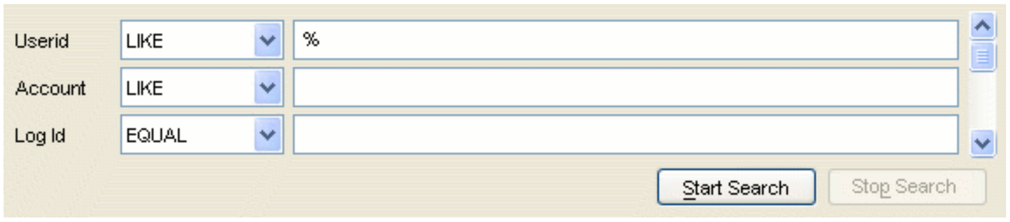


Figure 18. CMBSearchTemplateViewer bean

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 is 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.

You must match the query syntax for a free or boolean text search when you enter the query string in the text field (see the `DKDatastoreTS` class). For more information, see the *Application Programming Reference*.

Validating or editing fields of the CMBSearchTemplateViewer

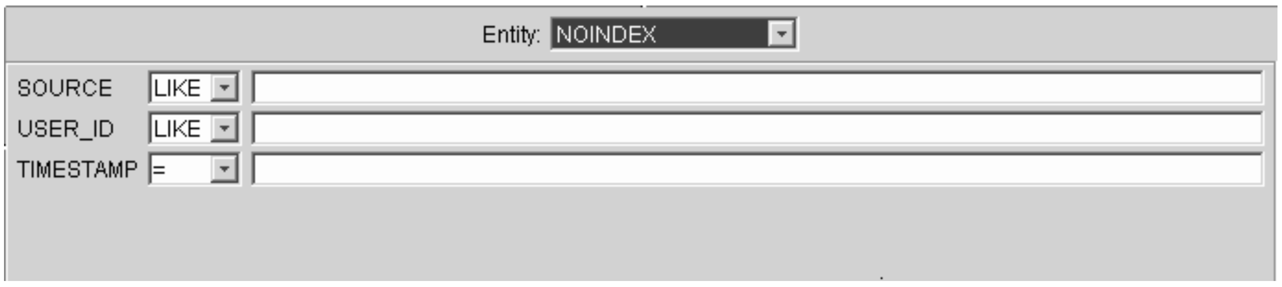
You can provide validation logic for the `CMBSearchTemplateViewer` bean to modify search criteria entered by the user.

Provide validation logic by using a handler for the CMBTemplateFieldChangedEvent. Before this event is called, the current values of the search criteria are stored in the CMBTemplate returned by the getTemplate method. You can then examine and change the criteria. After the event handling is complete, the new values are displayed.

CMBSearchPanel bean

For all servers, the search window 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 in which 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 must be entered on a separate line in the multi-line text area.


Entity: NOINDEX

SOURCE	LIKE	
USER_ID	LIKE	
TIMESTAMP	=	

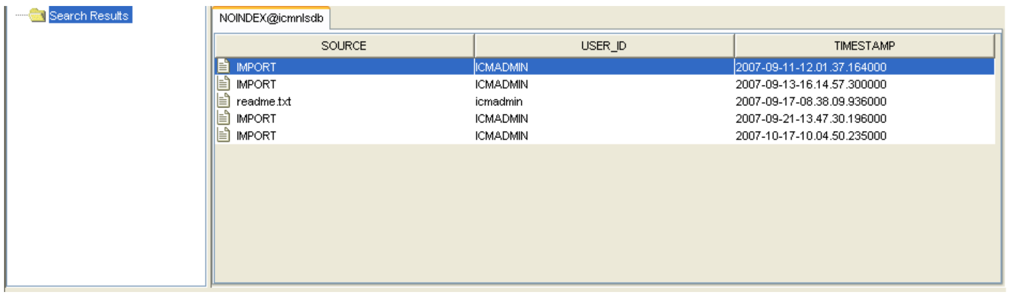
Figure 19. CMBSearchPanel Bean

CMBSearchResultsViewer bean

This bean displays search results. When the search result returns folders, use the 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.

The following screen capture shows the CMBSearchResultsViewer bean with the **Search Results** folder selected.



SOURCE	USER_ID	TIMESTAMP
IMPORT	ICMADMIN	2007-09-11-12.01.37.164000
IMPORT	ICMADMIN	2007-09-13-16.14.57.300000
readme.txt	icmadmin	2007-09-17-08.38.09.936000
IMPORT	ICMADMIN	2007-09-21-13.47.30.196000
IMPORT	ICMADMIN	2007-10-17-10.04.50.235000

Figure 20. CMBSearchResultsViewer bean

CMBSearchResultsViewer Tree pane
The tree pane (on the left) contains a main folder labeled **Search Results**. Beneath that folder is every 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 has child components, 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 attribute of the child component. 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 sort-options menu appears when you right-click on a table column heading. Click **Sort Ascending** to sort the items in the table in ascending order. Click **Sort Descending** to sort the items in descending order. Another pop-up menu appears when you right-click a folder other than the **Search Results** folder in the tree pane, or right-click a document or folder in the details pane. The menu lets you view folder details in the tree pane, or edit Attributes for folders.

Optional: Use the `CMBViewFolderEvent` to cause the `CMBFolderViewer` bean to display the contents of the selected folder instead of displaying the details of the folder within the `CMBSearchResultsViewer` bean.

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 menu, a `CMBItemActionEvent` occurs.

Overriding pop-up menus

You can override the pop-up menus on the `CMBSearchResultsViewer` and `CMBFolderViewer` using either a different menu or no menu.

To turn off the default menus, use `setDefaultPopupMenu(false)`.

When you right-click a folder in the tree pane, a `CMBFolderPopupEvent` is generated. When you right-click an item in the details pane, a `CMBItemPopupEvent` is generated. You can use a handler to provide a different 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. The following screen capture shows the tree and details panes of the `CMBFolderViewer` bean.

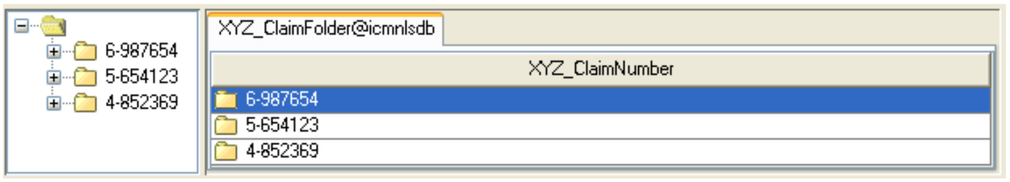


Figure 21. CMBFolderViewer bean

The `CMBFolderViewer` bean displays a tree of folders on the left pane. The right pane displays a notebook of tables of the documents contained by folder selected in the tree pane. A resizeable splitter separates the tree and notebook panes.

CMBFolderViewer Tree pane
: The tree pane contains folders. Nested folders appear beneath each folder.

CMBFolderViewer Details pane
: The details pane contains the contents of the folder that is selected in the tree pane. The contents display in a notebook with a tab for each entity (index class, item type, or other) and server, that the items in the table are indexed under. The tab names have the form: `index class @ server` where *index class* is the name of the index class and *server* is the name of the server. Within each notebook page is a table displaying the documents and folders within the selected folder. The table columns change to display the attributes according to the index class.

Pop-up menus
: The behavior of the pop-up menus for the folder viewer is identical to that of the search results viewer.

Double-click action
: Double-clicking in the folder viewer is identical to that of the search results viewer.

CMBDocumentViewer bean

Displays one or more documents by 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:

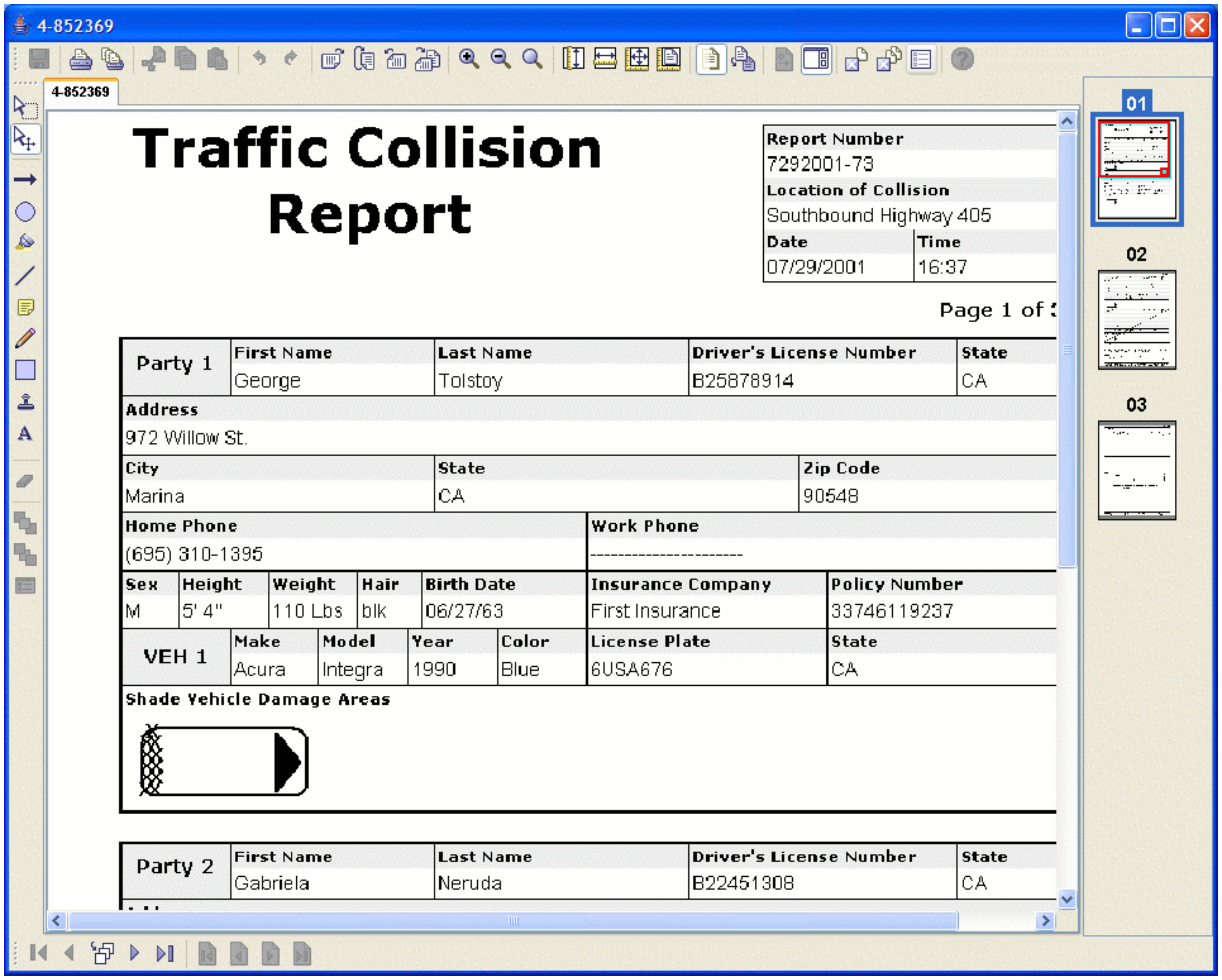
Java-based viewers
: These viewers must extend the class `CMBJavaDocumentViewer`.

Non-Java viewers
: Any executable program can 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 is displayed within the display region of the `CMBDocumentViewer` bean (for Java-based viewers only).

`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 monitor.

A call to `CMBDocumentViewer terminate()` waits until all document closed events are processed. If you call `terminate()` from within the document closed event handler, a deadlock might occur, causing the program to stop. To avoid this problem, when calling `terminate()` from within the `onDocumentClosed(CMBDocumentClosedEvent)` event handler, call the `CMBDocumentViewer.terminate()` method by using `SwingUtilities.invokeLater(Runnable)`. This technique adds the `terminate()` call to the end of the event queue and continues processing the other events in the queue (such as handling the other document closed events) before calling the terminate method.


4-852369

Traffic Collision Report

Report Number	7292001-73
Location of Collision	Southbound Highway 405
Date	07/29/2001
Time	16:37

Page 1 of

Party 1	First Name	Last Name	Driver's License Number	State
	George	Tolstoy	B25878914	CA

Address	972 Willow St.	
City	State	Zip Code
Marina	CA	90548
Home Phone	Work Phone	
(695) 310-1395	----------------------	

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 22. 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 with 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 is the Java class name, including the class suffix. For executable programs, the application name is the name of the program.
- Using the `Mime2App` property of `CMBDocumentViewer`. This property can be set to an instance of a properties object that maps MIME types to application names.

If a viewer is specified for a MIME type in both IBM Information Integrator for Content Administration and the `Mime2App` property, the viewer specified by the `Mime2App` property takes precedence.

Default viewers

If no viewer is specified for a particular content type, a default viewer is loaded.

For documents from Content Manager OnDemand, the Content Manager OnDemand client (in view-only mode) is launched, if installed. Documents from all other content servers are viewed by using the `CMBGenericDocViewer` from the Java viewer toolkit. To edit annotations, select **Edit Document** from the **File** menu of the viewer.

Launching external viewers

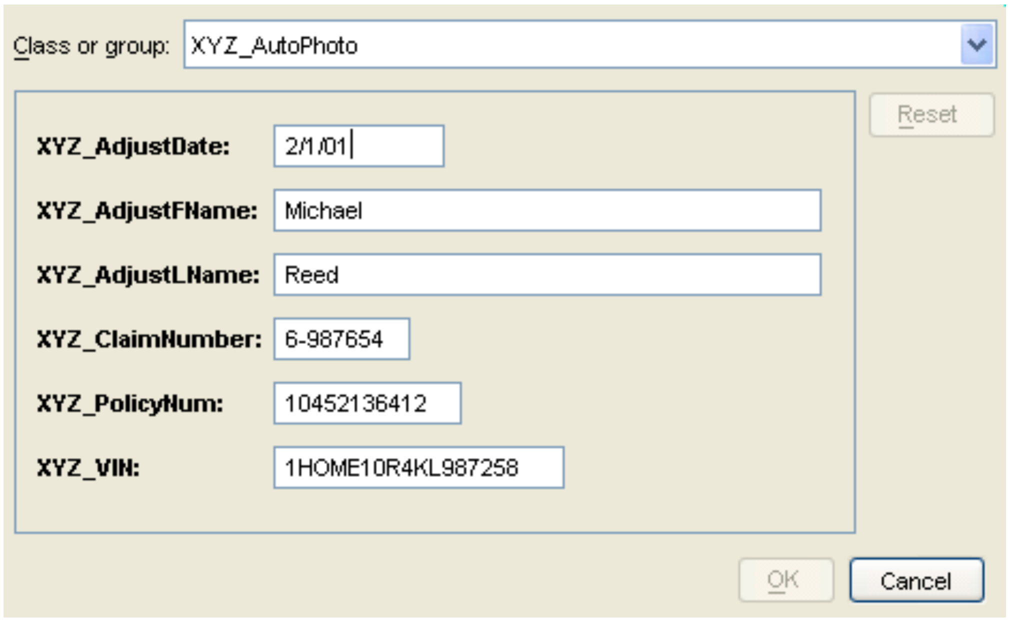
You can open external viewers by using the `Mime2App` property.

Use the `Mime2App` property of `CMBDocumentViewer` to specify applications to open 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 in which users can update the index class and indexing attributes for an item.

The `CMBItemAttributesEditor` bean displays a window for viewing and modifying the entity (item type) and attributes of a folder or document.



Class or group:	XYZ_AutoPhoto
XYZ_AdjustDate:	2/1/01
XYZ_AdjustFName:	Michael
XYZ_AdjustLName:	Reed
XYZ_ClaimNumber:	6-987654
XYZ_PolicyNum:	10452136412
XYZ_VIN:	1HOME10R4KL987258

Figure 23. 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 on) initially contain the current values for the item.

If a user selects 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

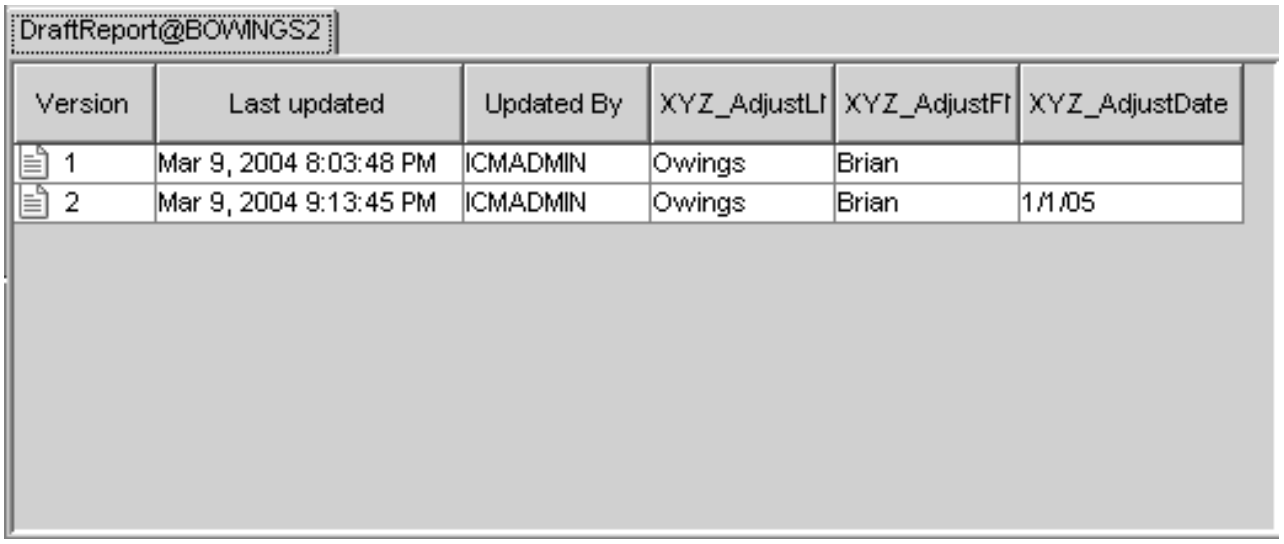
You can provide additional validation logic to the `CMBItemAttributesEditor` that verifies attribute values entered by the user. It modifies them or rejects an update if the values are not valid.

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_AdjustFt	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 24. CMBVersions Viewer bean

General behaviors for visual beans

The following sections describe properties and behaviors that are common among visual beans.

“Properties” on page 458

“Save/restore configuration” on page 458

“Support for child component attributes” on page 458

“Help events” on page 458

Properties

The three properties shared by visual beans are connection, collation strength, and hiding or showing of buttons.

Visual beans have the following three properties:

Connection
: Each bean has a `Connection` property, which refers to an instance of the `CMBConnection` non-visual bean. You must set the `Connection` property for the visual bean to operate correctly.

CollationStrength
: All beans that perform sorting have a `CollationStrength` property. The values defined for `CollationStrength` property are the same values defined for the `java.text.Collator` class of Java.

Hiding/Showing buttons
: You can hide or show the buttons that appear on all visual beans. Use the `setNameButtonVisible` property, where **Name** is the name of the button.

Save/restore configuration

To save and restore fields, use the `loadConfiguration` and `saveConfiguration` methods.

The `CMBSearchTemplateViewer`, `CMBSearchResultsViewer`, and `CMBFolderViewer` classes have two methods, `loadConfiguration` and `saveConfiguration`, that you can use to save and restore field values and column sizes between application sessions. Both of these methods take a properties object as an argument. You can use the same properties object for all three beans, but 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 behavior is like that of the Windows client. For details, see the Windows client documentation.

Help events

Each visual bean generates a `CMBHelp` event when a user requests help, either by clicking the **Help** button or pressing F1.

The following beans generate help-related events when a user presses F1 or the **Help** button from a secondary window:

CMBChangePasswordHelpEvent
: **Help** is clicked in the Change Password window

CMBUpdateMappingHelpEvent
: **Help** is clicked in the Update Mapping window

CMBLoginFailedHelpEvent
: **Help** is clicked in the Server Logon Failed window

CMBServerUnavailableHelpEvent
: **Help** is clicked in the Server Unavailable window

"Tip"

Tip:

One way of handling help from all these sources is to create a single class that implements the listeners for all the events.

Within the onHelp method, you might need additional logic to determine which bean is the source of the event and to 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 must implement the handlers for the events of the visual bean it is replacing. In addition, it must at least generate the key events of the bean it is replacing. The key events are described in the following table.

Table 55. 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 the data needed for implementing the bean function is available either from events that the bean is handling or from the CMBConnection non-visual bean.

Building an application by using visual beans

The visual beans fit together when you build an application.

A sample client application that was written by using the visual beans is provided. The source files for the sample are located in: *%IBMCMROOT%*/samples/java/beans/gui.. For details about the sample client and setup requirements, see the readme.html file in this directory.

"Connecting the visual beans"

"Using beans in more than one window or dialog" on page 461

Connecting the visual beans

One scenario for connecting visual beans to create a simple application is by using a target bean as listener when connecting all the beans.

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 the following figure, the lines from each of the beans to the connection bean indicate that the bean contains a reference to the connection bean. This reference is created by setting the connection property for each bean. For example, to create a reference from the logon window bean to the connection bean, a line of code is needed.

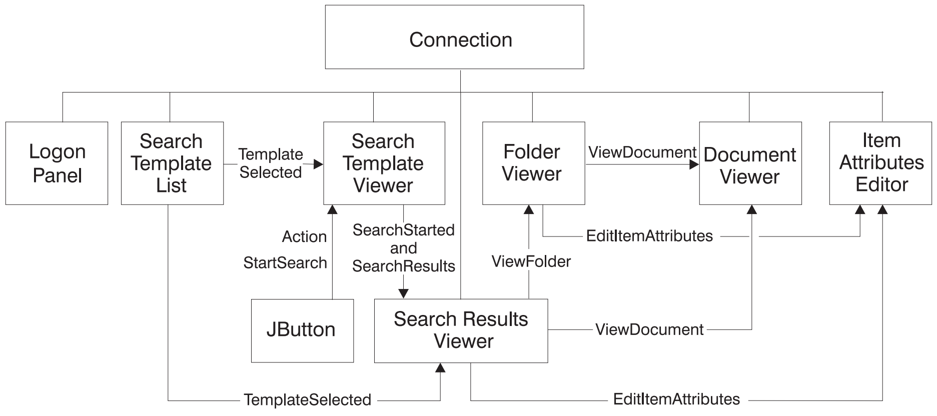


Figure 25. Visual bean connections

The previous figure shows nine beans. A `JFrame` or other container bean is the parent of all these beans. The following sequence shows one possible order of events during run time:

1. Enter a user ID and password in the logon window and click **OK**. The `CMBLogonPanel` bean invokes the `connect` method of the `CMBConnection` bean to establish the connection to the server. After 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 called. The `CMBSearchTemplateList` bean listens to the appropriate events of the `CMBConnection` bean. `CMBSearchTemplateList` sets up the listeners when a `CMBConnection` bean is associated with it by 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 listen 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 the view, select the attributes of a document or folder 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, 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.

A window is typically displayed when an event is sent. 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 IBM 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 IBM Information Integrator for Content search template and server configuration. This uses an XML schema called `cmadmin.xsd` (located in *IBMCMROOT*`/config/`) to represent the IBM Content Manager Version 8 administration objects in an XML file.
- Data model metadata objects which define the structure for item types, component types, and IBM 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 IBM 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 IBM Content Manager programming layers”
“Importing and exporting IBM Content Manager metadata by using XML services” on page 466
“Importing and exporting IBM Content Manager data instance objects as XML” on page 488
“Importing and exporting XML object dependencies” on page 491
“Extracting content from different XML sources” on page 491
“Hierarchical item operations using XML services” on page 492
“Migrating hierarchical items with XML services” on page 493
“Mapping a user-defined schema to a storage schema with the XML schema mapping tool” on page 494
“Programming run-time operations through the XML JavaBeans” on page 496

How XML services work with other IBM Content Manager programming layers

Certain programming layers require converted XML objects to work with the IBM Content Manager connector.

The layers include:

Web services
IBM 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 IBM Content Manager connector XML APIs and the IBM 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 IBM Content Manager supports.

IBM Content Manager connector (XML)
XML application programming interfaces that can import and export data model metadata objects, administrative metadata objects, and data instance objects.

The following figure illustrates how these XML layers relate to the IBM Content Manager connector.

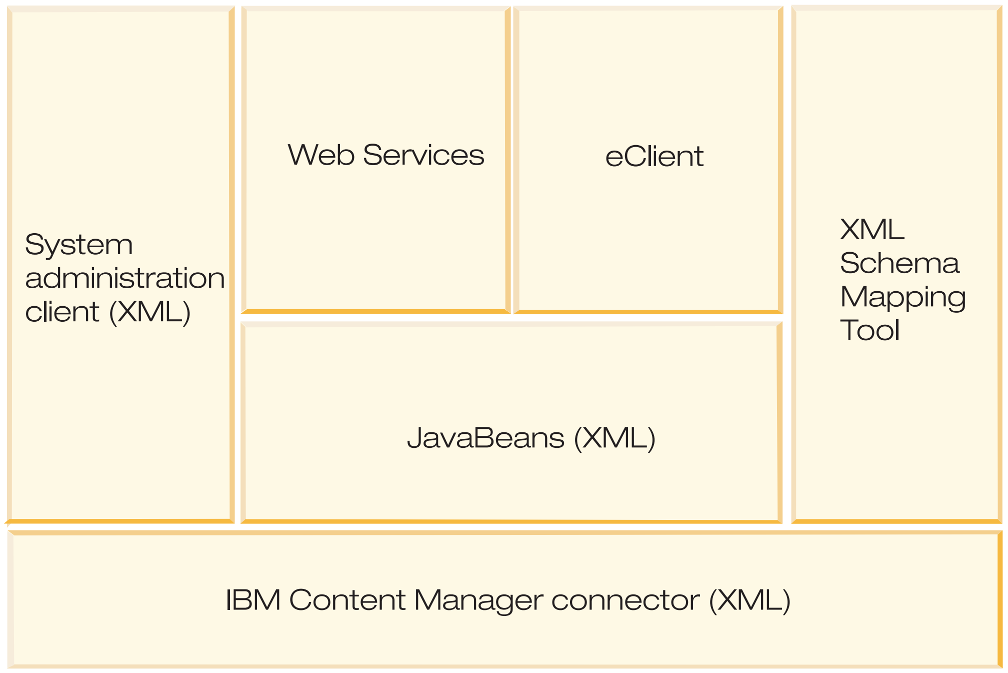

System administration client (XML)
Web Services
eClient
XML Schema Mapping Tool
JavaBeans (XML)
IBM Content Manager connector (XML)

Figure 26. XML services programming layers

The following flowchart depicts a real-world scenario for communicating with IBM Content Manager through its XML interface:

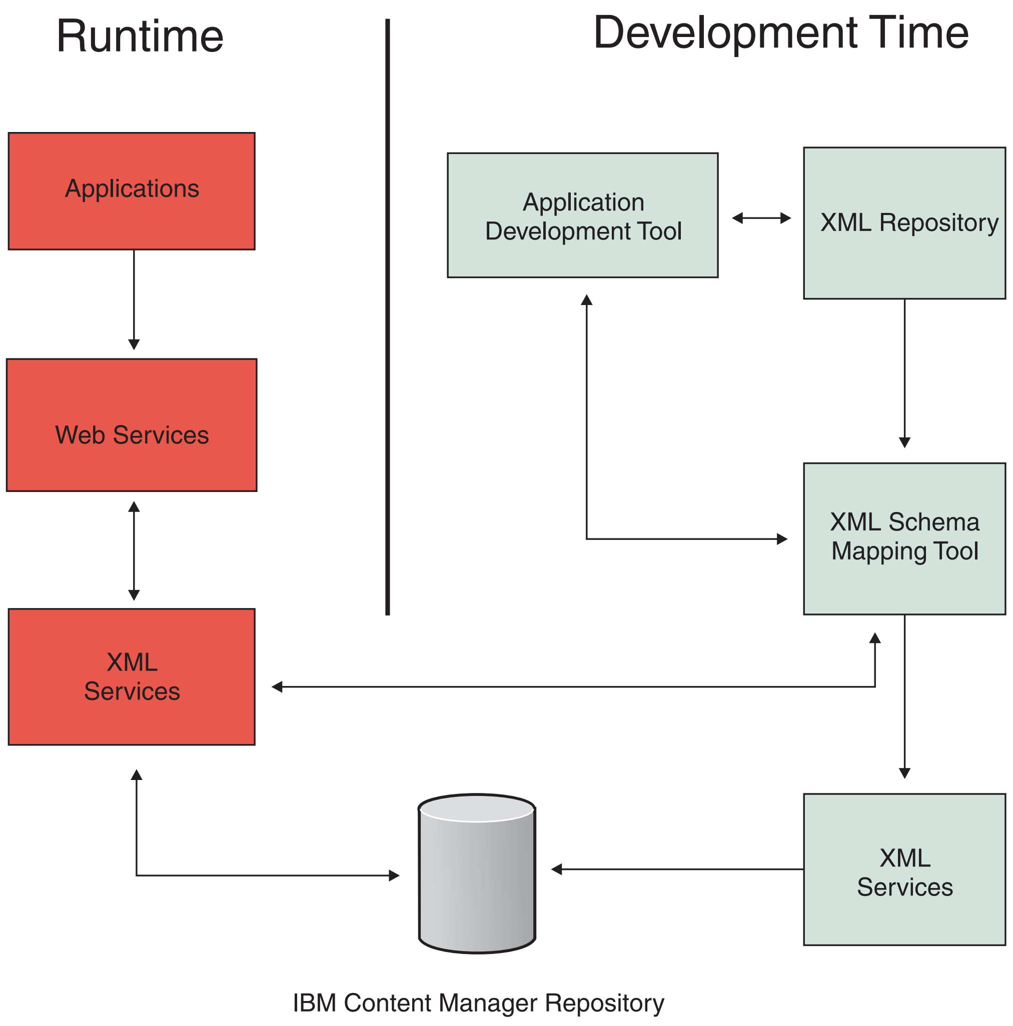


Figure 27. XML services communication flowchart

1. The XYZ insurance company defines schemas to hold its data, by using application development tools like WebSphere Application Developer (WSAD) to create them. The company also chooses to store its data in IBM Content Manager as items.
2. In an effort to migrate all data to IBM Content Manager, the XYZ insurance company uses the IBM Content Manager XML schema mapping tool to convert its schema to ones that IBM 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 IBM 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 IBM 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 IBM 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 IBM Content Manager metadata by using XML services

You can import and export two types of metadata by using the Content Manager APIs.

The types of metadata that you can import and export by using the Content Manager APIs are:

Administrative objects
: Users, user groups, resource manager configuration, document routing definitions, ACL privileges, and the IBM Information Integrator for Content search template and server configuration. An XML schema called `cmadmin.xsd` (located in *IBMCMROOT*/`config/`) is used to define the XML files containing IBM Content Manager Version 8 administration objects.

Data model objects
: The structure of item types, component types, and IBM 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 IBM Content Manager metadata as intermediate files, you can program a number of scenarios:
- Customizing an application to administer and update data in IBM Content Manager through the XML interface.
- Transferring metadata from one IBM Content Manager system to another IBM Content Manager system (considering any data conflicts).
- Transferring entities, search templates, and server configuration from one IBM Information Integrator for Content system to anotherIBM Information Integrator for Content system (considering 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 IBM 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 IBM 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 (by 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 do not specify or do not OR this constant, then the import process stops on any error.

DK_CM_XML_IMPORT_CREATE_UPDATE
: If you specify this constant, then the `ingest()` method replaces objects that exist. If you do not specify this constant, then the import fails with an error for any object definitions that 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.

“Importing and exporting administration objects as XML”

“Importing and exporting IBM Content Manager data model objects as XML schema files (XSD)” on page 469

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 IBM 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 IBM 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 IBM 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 IBM Content Manager server:

```
<?xml version="1.0" encoding="UTF-8"?>
<CMSystemAdminDefinitions
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>
</CMSystemAdminDefinitions>
```

Importing and exporting IBM Content Manager data model objects as XML schema files (XSD)

The XML instance service class, `DKXMLSysAdminService`, contains two methods for importing and exporting IBM 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 IBM 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 _x*YYYY* , 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.

IBM Content Manager defines the storage schema by using the following steps:

1. IBM Content Manager attempts to map any available construct or feature in the XML schema to an IBM Content Manager data model concept, as shown in the following table.
2. If no IBM 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 the following table.

3. IBM Content Manager instances (for example, item-level ACLs and semantic types) are represented as XML elements (in the IBM Content Manager namespace) imported from the `cmdatamodel.xsd` file. These are shown in the following table.

Table 56. How IBM 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, and so on.
--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 might 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				

Table 56. How IBM Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-Variable character			This represents the derived string simple type definition with the minLength and maxLength constraints. For example, <simpleType name="varchar_0,256"> <restriction base='string'> <minLength value="0"/> <maxLength value="256"/> </restriction> </simpleType>	The minimum and max length can be specified.
--alphabetic				
--numeric				
--alphanumeric				
--Extended alphanumeric				
--other				
-Short integer			This represents the short built-in derived type.	This is a specific min and max value.
-Long integer			This represents the integer built-in derived type.	This is a specific min and max value.
-Double			This represents the double built-in primitive type.	
-Decimal			This represents the decimal built-in primitive type.	This is a specific length and fixed places.
-Date			This represents the date built-in primitive type with facet value conversion (convert into the facet format which is understood by Content Manager and DB2).	
-Time			This represents the time built-in primitive type.	
-Timestamp			This represents the dateTime built-in primitive type with facet value conversion.	
-BLOB			A regular string simple type with a CM meta-attribute namespace. For example: <attribute name=".." type="base64Binary"> <annotation><appinfo> <CM:dataType value="BLOB"/> </appinfo> </annotation> </attribute>	The length can be specified.
-CLOB			This maps to a regular string simple type with a CM meta-attribute namespace. For example: <attribute name="..." type="string"> <annotation><appinfo> <CM:datatype value="CLOB"/> </appinfo></annotation> </attribute>	The length can be specified.
-max	integer		The maxInclusive element in the dervied simple type definition. For example: <simpleType name='integer_max100'> <restriction base='integer'> <maxInclusive value="100"/> </restriction> </simpleType> This creates a new type. IBM Content Manager associates a property with the attribute.	This is for short and integers only.

Table 56. How IBM Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-min	integer		The minInclusive element in the derived simple type definition. For example: <simpleType name='integer_min10' > <restriction base='integer'> <minInclusive value="10" /> </restriction> </simpleType>	This is for short and integer only.
-length	integer		The length element in the derived simple type definition. For example: <simpleType name='char_256'> <restriction base="string"> <length value="256"/> </restriction> </simpleType>	
-scale/precision	integer		The fractionDigits and totalDigits elements in the derived simple type definition. For example: <simpleType name='decimal_8,3'> <restriction base='decimal'> <fractionDigits value="3"/> <totalDigits value="8"/> </restriction> </simpleType>	This is for decimals only.
Attribute groups (global)			The attributeGroup element in the XSD. For example: <xs:attributeGroup name="myAttrGrp"> <xs:attribute .../> ... </xs:attributeGroup>	
-Name	string	Yes	The name attribute in the attributeGroup element. For example: <xs:attributeGroup name="myAttrGrp"> <xs:attribute .../> ... </xs:attributeGroup>	
-Attributes	array of string	yes	The child elements of the attributeGroup element. For example, <xs:attributeGroup name="myAttrGrp"> <xs:attribute .../> ... </xs:attributeGroup> The attribute name has 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, myAttrGrp.myStringAttr.	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 xs:schema element as the parent. For example: <xs:schema ....> <xs:element name="elem1"> .... </xs:element> </xs:schema> It also has a similar XML Schema structure as the "Component Type Definition" because an item type is a component type based on the IBM Content Manager data model.	

Table 56. How IBM Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-Name	string	yes	The name attribute in the xs:element declaration. For example: <xs:schema ....> <xs:element name="elem1"> .... </xs:element> </xs:schema>	
-Properties			Add an optional element definition right under the root element for describing any instance-level property. It refers the element defined in the cmdatamodel.xsd schema file. For example: <element ref="cm:properties" minOccurs="0" maxOccurs="unbounded">	Use in the item instance to describe any instance-level properties associated with the item, such as semantic type, ACL, creation time, and so on.
-Resource Object			Add an optional element definition right under the root element for describing any resource content associated with the item. It refers 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 refers 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 • Non-resource item • Resource item • 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 includes 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			

Table 56. How IBM Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-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 be referred only by a document type. No XML instance can be based on this document type.	
Component Type:			This maps to an element declaration in XML schema. If the definition exists in a standalone fashion (it means it does not associate with any itemType), it maps to a root element declaration in the XML schema with a cm:entityType element with value="component". Mostly non-root element--children of another element. This would be root element if there is no item type definition it corresponds to. This accepts only "sequence" , but not "all", "choice" and nested particle. The IBM 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 includes the name: <component type name>.<attribute group name>. The global attribute group definition includes 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 uses all the type definition and annotations described in the global attribute, which applies to local.	
-Is required	boolean		This represents the used attribute name in the attribute declaration. For example: <xsd:attribute name=".." use="required" .../> Here is how you map the value of nullable attribute to the "use" and "default" attribute. "nullable --> use="optional" "non-nullable w/o default --> use="required" "non-nullable w/ default --> use="optional" default="X"	
-Max value	integer		This defines a new simpleType with a new max value, such as the maxInclusive element in the derived simple type definition. For example: <simpleType name='integer_max100'> <restriction base='integer'> <maxInclusive value="100"/> </restriction> </simpleType>	

Table 56. How IBM Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
-Min value	integer		This represents the minInclusive element in the derived simple type definition. For example: <simpleType name='integer_min10'> <restriction base='integer'> <minInclusive value="10"/> </restriction> </simpleType>	
-Default value	string		This represents the default attribute in the attribute declaration. For example: <xsd:attribute name=".." default ="10" ... />	
Sub-components / child components			This maps to a child element declaration (by using sequence) with the min and max occurrence of a child component mapped to the minOccur and maxOccur 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 the annotations marked only 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 has 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 be created only through the Federated connector.	
-name	string	yes	The name attribute in the xs:element declaration. For example: <xs:schema ....> <xs:element name="elem1"> <xs:annotation><xs:appinfo> <xs:entityType value="federated"/> </xs:appinfo></ xs:annotation> .... </xs:element> </xs:schema>	
-Text searchable	boolean			
-Enabled creating the native federated folder	boolean			
-Attribute:			This maps to an XSD attribute declaration within the element.	
--name	string		The name attribute in the xs:attribute declaration. For example: <xs:schema ....> <xs:element name="elem1"> <xs:annotation><xs:appinfo> <xs:entityType value="federated"/> </xs:appinfo></ xs:annotation> <xs:complexType> <xs:attribute name="attr1"...> .... </xs:element> </xs:schema>	

Table 56. How IBM Content Manager data model objects map to the storage schema constructs (continued)

Object	Data type	Required	Corresponding storage schema construct	Comments
--datatype and other attribute information, such as length, precision, scale, max, min, nullable, queryable, writeable			Use the same definition as in the attribute category, but some representation is not there (such as the text-searchable information).	
-Fed entity name	string			
-Fed attr name	string			
-Native server name	string			
-Native server type	string			
-Native entity name	string			

For Table 57, 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 57. How IBM Content Manager data model objects map to annotations in the storage schema

Object	Information specified in annotation	Scope of the annotation	Instance-level information
Attributes (global)			
-Type (possible type as follows)			
--Character:			
--alphabetic	You use the element cm:stringType with a value attribute in enumeration type {ALPHA, NUMERIC, ALPHANUMERIC, ALPHANUMERIC_EXT, OTHER}, to represent the value. Putting the pattern element is optional. In this case, <cm:stringType value="ALPHA"/>	GLOBAL, LOCAL, VIEW	
--numeric	<cm:stringType value="NUMERIC"/>	GLOBAL, LOCAL, VIEW	
--alphanumeric	<cm:stringType value="ALPHANUMERIC"/>	GLOBAL, LOCAL, VIEW	

Table 57. How IBM Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
--Extended alphanumeric	<cm:stringType value="ALPHANUMERIC_EXT"/>	GLOBAL, LOCAL, VIEW	
--other	<cm:stringType value="OTHER"/>	GLOBAL, LOCAL, VIEW	
-Variable character			
--alphabetic	Same as in the Character case <cm:stringType value="ALPHA"/>	GLOBAL, LOCAL, VIEW	
--numeric	<cm:stringType value="NUMERIC"/>	GLOBAL, LOCAL, VIEW	
--alphanumeric	<cm:stringType value="ALPHANUMERIC"/>	GLOBAL, LOCAL, VIEW	
--Extended alphanumeric	<cm:stringType value="ALPHANUMERIC_EXT"/>	GLOBAL, LOCAL, VIEW	
--other	<cm:stringType value="OTHER"/>	GLOBAL, LOCAL, VIEW	
-Short integer			
-Long integer			
-Double			
-Decimal			
-Date			
-Time			
-Timestamp			
-BLOB	<CM:dataType value="BLOB"/>	GLOBAL, LOCAL, VIEW	
-CLOB	<CM:dataType value="CLOB"/>	GLOBAL, LOCAL, VIEW	
-max			
-min			
-length			
-scale/precision			
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
Attribute groups (global)			
-Name			

Table 57. How IBM Content Manager data model objects map to annotations in the storage schema (continued)

Object	Information specified in annotation	Scope of the annotation	Instance-level information
-Attributes			
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
Reference attribute	Indicated by an annotation element, cm:referenceAttribute with the following attributes:	GLOBAL, LOCAL, VIEW	
-Name			
-Reference delete rule	{cm:referenceAttribute} cm:deleteRule=<string> in enumeration type {NO_ACTION, SET_NULL, CASCADE, RESTRICT.	GLOBAL, LOCAL, VIEW	
-Reference sequence number	{cm:referenceAttribute} cm:sequenceNumber=<short>	GLOBAL, LOCAL, VIEW	
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
Item Type: definition			
-Name			
-Description w/ lang code	cm:description element with a value attribute and xsi:lang attribute (to indicate which language it belongs to).	GLOBAL	
-New version policy	cm:versionPolicy element with a value attribute in enumeration type {NEVER, ALWAYS, BY_APPLICATION}.	GLOBAL	
-Maximum total versions	cm:maximumVersions element with a value attribute.	GLOBAL	
-Properties			The cmdatamodel.xsd file contains the definition of the properties.
-Resource Object			The cmdatamodel.xsd3 file contains the definition of the resourceObj.

Table 57. How IBM 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 is part of the predefined IBM 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 is in the item type level and the element is 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 57. How IBM 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} 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 <element name="cm:resourceObject"> <complexType> <choice> <element name="content"> <complexType> <attribute name="value" type="base64Binary"/> </complexType> </element> <element name="url"> <complexType> <attribute name="value" type="anyURI"/> </complexType> </element> </complexType> </element> Note that the type would be either "base64Binary" or "hexBinary". This is part of the predefined IBM 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 predefined ICMPARTs (ICMANNOTATION, ICMBASE, ICMBASETEXT, ICMBASESTREAM, ICMNOTELOG). They are handled or represented similarly to the regular user part, which is 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" ...> <xsd:annotation> <xsd:appinfo> <cm:ACL name="..."/>	GLOBAL	
-RM name	cm:RM element with name and attributes under the reference element name.	GLOBAL	
-SMS coll name	cm:SMSCollection element with name and attributes under the reference element name.	GLOBAL	
-New version policy	cm:versionPolicy element name with a value attribute in enumeration type {NEVER, ALWAYS, BY_APPLICATION}.	GLOBAL	
4. Document part type	A cm:entityType element with the value="PART" to indicate this element is a document part type.	GLOBAL, VIEW	If it is an IBM Content Manager predefined PART type, it is located in the cmdatamodel.xsd file.
-XDO class name	cm:XDOClass element with a value attribute.	GLOBAL	

Table 57. How IBM 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} sourceItemType=<string>	GLOBAL	
-Item type to be linked	{cm:autoLinkAttributes} cm:targetItemType=<string	GLOBAL	
-Auto linking attribute	{cm:autoLinkAttributes} cm:attributeName=<string>	GLOBAL	
-Auto linking attribute	{cm:autoLinkAttributes} cm:attributeGroupName=<string>	GLOBAL	
-Link type	{cm:autoLinkAttributes} cm:autoLinkType=<string>	GLOBAL	
-Auto Linking SMS	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} cm:constraintName=<string>	GLOBAL	
-Update rule	{cm:foreignKey} cm:updateRule={RESTRICT, NO_ACTION}	GLOBAL	

Table 57. How IBM 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} cm:deleteRule={RESTRICT, CASCADE, NO_ACTION, SET_NULL}	GLOBAL	
-Source component	{cm:foreignKey} cm:sourceComponent=<string>	GLOBAL	
-Target item type	{cm:foreignKey} cm:targetItemType=<string>	GLOBAL	
-Target external table	{cm:foreignKey} cm:targetTable=<string>	GLOBAL	
-Attribute Pairs	A cm:attributePair element under the cm:foreignKey element with the following attributes:	GLOBAL	
-Source attribute group	{cm:attributePair} cm:sourceAttributeGroup=<string>	GLOBAL	
-Source attribute	{cm:attributePair} cm:sourceAttribute=<string>	GLOBAL	
-Target attribute group	{cm:attributePair} cm:targetAttributeGroup=<string>	GLOBAL	
-Target attribute	{cm:attributePair} cm:targetAttribute=<string>	GLOBAL	
-External table column name	{cm:attributePair} cm:targetTableColumn=<string>	GLOBAL	
User exits	cm:userExit element with the following attributes:	GLOBAL, VIEW	
-Exit name	{cm:userExit} cm:name=<string>	GLOBAL, VIEW	
-Function name	{cm:userExit} cm:functionName=<string>	GLOBAL, VIEW	
-DLL name	{cm:userExit} cm:DLLName=<string>	GLOBAL, VIEW	
Access Module	A cm:accessModule element with the following attributes:	GLOBAL	
-name	{cm:accessModule} cm:name=<string>	GLOBAL	
-result	{cm:accessModule} cm:result=<integer>	GLOBAL	
-status	{cm:accessModule} cm:status=<short>	GLOBAL	
-version	{cm:accessModule} cm:version=<short>	GLOBAL	
Previous Access Module	A cm:previousAccessModule element with a version attribute.	GLOBAL	
Component Type:			
-Attribute groups			
-Reference attributes			

Table 57. How IBM 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, LOCAL, VIEW	
-Text Index Information	A cm:textIndexInfo element, with the following attributes:.	GLOBAL, LOCAL	
-Commit count	{cm:textIndexInfo } cm:commitCount=<integer>	GLOBAL, LOCAL	
-Format	{cm:textIndexInfo } cm:format=<integer>	GLOBAL, LOCAL	
-Index CCSID	{cm:textIndexInfo } cm:CCSID=<integer>	GLOBAL, LOCAL	
-Index Directory	{cm:textIndexInfo } cm:directory=<string>	GLOBAL, LOCAL	
-Index Language Code	{cm:textIndexInfo } cm:langCode=<string>	GLOBAL, LOCAL	
-Minimum changes	{cm:textIndexInfo } cm:minChanges=<integer>	GLOBAL, LOCAL	
-Model File	{cm:textIndexInfo } cm:modelFile=<string>	GLOBAL, LOCAL	
-Model Name	{cm:textIndexInfo } cm:modelName=<string>	GLOBAL, LOCAL	
-Model CCSID	{cm:textIndexInfo } cm:modelCCSID=<integer>	GLOBAL, LOCAL	
-UDF Name	{cm:textIndexInfo } cm:UDFName=<string>	GLOBAL, LOCAL	
-UDF Schema	{cm:textIndexInfo } cm:UDFSchema=<integer>	GLOBAL, LOCAL	
-Update Frequency	{cm:textIndexInfo } cm:updateFrequency=<string>	GLOBAL, LOCAL	
-Update Frequency Unit	{cm:textIndexInfo } cm:updateFrequencyUnit= <{MINUTE, HOUR}>	GLOBAL, LOCAL	
-Working Directory	{cm:textIndexInfo } cm:workingDir=<string>	GLOBAL, LOCAL	
-Is representing item	cm:representative element with a value attribute in boolean type.	LOCAL	
-Is unique	cm:unique element with a value attribute in boolean type.	LOCAL	
-Is required			
-Max value			
-Min value			
-Default value			
-Resource Manager attribute	cm:isResourceManagerAttr element with a value attribute in boolean type.	LOCAL	

Table 57. How IBM 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, LOCAL	
--Name	{cm:databaseIndexInfo } cm:name=<string>	GLOBAL, LOCAL	
--Unique	{cm:databaseIndexInfo } cm:unique=<boolean>	GLOBAL, LOCAL	
--Index Schema	{cm:databaseIndexInfo } cm:indexSchema=<string>	GLOBAL, LOCAL	
--Attributes	A subelement, cm:indexedAttribute element under the same cm:databaseIndexInfo element with the following attributes:	GLOBAL, LOCAL	
---Name	{cm:indexAttribute } cm:name=<string>	GLOBAL, LOCAL	
---Index Order	{cm:indexAttribute } cm:indexOrder={ASCENDING, DESCENDING}	GLOBAL, LOCAL	
Delete rule	A cm:deleteRule element with a value attribute {RESTRICT, CASCADE}.	LOCAL	
Sub-components/ child components			
Item Type View:	A cm:entityType element with the value="ITEM" and a cm:entityView element with cm:baseEntityType attribute to indicate which item type this refers to.	VIEW	
Component Type View:			
-Attributes	cm:representative element with a value attribute in boolean type	VIEW	
--readable	cm:readable element with a value attribute in boolean type. No such concept exists in the XML schema. Only has "fixed" in term of the value, or "prohibited" in term of attribute name definition.	VIEW	
--writable	cm:writeable element with a value attribute in boolean type	VIEW	
--queryable	cm:queryable element with a value attribute in boolean type	VIEW	
--excludeRow	cm:excludeRow element with a value attribute in boolean type	VIEW	
--View compare value	cm:viewCompareValue element with a value attribute	VIEW	
--View operator	cm:viewOperator element with a value attribute in enumeration type {OPCODE_EQ}	VIEW	

Table 57. How IBM 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 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 IBM Content Manager storage schemas"

"Constraints for converting to IBM Content Manager storage schemas" on page 487

Unsupported XML types in the IBM Content Manager storage schemas

There are some XML types that are not supported in IBM Content Manager storage schemas.

The IBM Content Manager storage schemas do not support the following primitive data types:

- string (must be associated with either length properties)
- Boolean
- duration
- gYearMonth
- gYear
- gMonthDay
- gDay
- gMonth
- hexBinary (supports base64Binary only)
- QName
- NOTATION

The IBM 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 IBM Content Manager storage schemas

The following constraints also apply to the IBM 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 IBM 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 by 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>
```

"Exporting IBM Content Manager DDO items as XML items" on page 489

"Importing XML items as IBM Content Manager DDO items" on page 490

Related information

Importing and exporting IBM Content Manager data model objects as XML schema files (XSD)

Exporting IBM Content Manager DDO items as XML items

The `extract()` method in `DKXMLDataInstanceService` converts a DDO (including all child component DDOs, links, and references) into a `DKXMLItem` object.

This `DKXMLItem` object contains the following data:

- XML document that represents the item version, including all child components.
- Properties, including system attributes, resource attributes, and links information.
- Any binary resource part content.

  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 IBM 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 IBM Content Manager DDO items

The ingest method in DKXMLDataInstanceService converts a DKXMLItem object into a DDO on the IBM 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 IBM Content Manager.

The new DDO belongs to an IBM Content Manager Version 8 item type or an earlier IBM Content Manager index class and can be stored only in IBM 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 in mind the following facts:

- You can import only into IBM Content Manager or earlier IBM 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 enter a DKXMLStreamItem, the 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 the ingest method 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 enters both an XML item XMLFile and its properties (both system and resource in a separate file XMLProperties) as input streams; and returns an IBM 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 IBM Content Manager data instance objects as XML

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 exports only 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 the data model dependency objects.

DK_CM_XML_EXPORT_SA_ONLY_PREREQUISITE
Exports only the administrative dependency objects (including authorization, authentication, and library server configuration).

DK_CM_XML_EXPORT_RM_ONLY_PREREQUISITE
Exports only the resource manager configuration (in the library server side) dependency objects.

DK_CM_XML_EXPORT_DR_ONLY_PREREQUISITE
Exports only the 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;
```

Hierarchical item operations using XML services

Content Manager EE XML services supports hierarchical item operations.

Creating a hierarchical item

Additional XML code is not required to set the ICM$NAME attribute when you create a hierarchical item. An optional <cm:parentFolder> element of the <cm:properties> element specifies the PIDString of the parent folder. The <cm:parentFolder> element is optional because you are not required to specify a hierarchical parent folder when a hierarchical item is created; a hierarchical item can be placed in the system default folder.

The system default folder, which is not enabled by default, is used only when the repository is configured to file all hierarchical items in a single system default folder. You can use the system default folder to enable an application that would otherwise not work by allowing it to use hierarchical items in a non-hierarchical way.

Updating a hierarchical item

Additional XML code is not required to change the ICM$NAME attribute when you update a hierarchical item. Because the hierarchical parent cannot be changed by setting DDO properties, the XML API does not support updating the hierarchical parent. If the <cm:parentFolder> element appears in the <cm:properties> element in an ingest that precedes an update, the API generates an error because this property is supported only when you create a DDO.

Using the extract method on a hierarchical item

For a DDO extract using the `DKXMLDataInstanceService` `extract` method, the hierarchical name appears automatically in the XML code of the hierarchical item because it is modeled as a user-defined attribute. The following example shows how the hierarchical name appears in the XML code of the items:

```
<ITEMTYPE ATTR1="the attr" ATTR2="the attr" ICM_x0024_NAME="hierarchicalName">
...
</ITEMTYPE>
```

Because the hierarchical parent folder property is not populated at retrieve time, it is not guaranteed to appear in the <cm:properties> element. If the hierarchical parent folder property is populated in the DDO when `extract` is called, or if the hierarchical parent folder link is retrieved in the DDO, then the <cm:parentFolder> element appears in the <cm:properties> element as follows:

```
<cm:properties type="item">
<cm:lastChangeUserid value="ICMADMIN"/>
<cm:lastChangeTime value="2010-07-14T22:58:21.734"/>
<cm:createUserid value="ICMADMIN"/
<cm:createTime value="2010-07-14T22:58:21.734"/>
<cm:semanticType value="4"/>
<cm:ACL name="PublicReadACL"/>
<cm:lastOperation name="RETRIEVE" value="SUCCESS"/>
<cm:parentFolder value="PIDString"/>
</cm:properties>
```

The hierarchical parent folder, which is treated the same in XML code as any other parent folders from any item type, appears in the XML code of a DDO only if the inbound link folder sources are retrieved into the DDO prior to XML extract. Similarly, hierarchical children items, which are treated the same in XML code as non-hierarchical children items, appear in the XML code only if outbound links are retrieved into the DDO. Hierarchical parents and non-hierarchical `DKFolder` parents appear in the XML code as follows:

```
<cm:folderSources>
<parentItem value="PIDString"/>
</cm:folderSources>
```

Hierarchical children and non-hierarchical children in the `DKFolder` collection appear in the XML code as follows:

```
<cm:folderContents>
<childItem value="PIDString"/>
</cm:folderContents>
```

Related information

Working with hierarchical item types

Migrating hierarchical items with XML services

When using the `extract` and `ingest` methods from `DKXMLDataInstanceService` to duplicate data from one system to another system, the XML ingest code does not automatically resolve and reconstruct the links and folder relationships within exported hierarchical items. Instead, you must intervene in the ingest process by modifying the XML code to manually reconstruct the folder and link relationships.

To manually reconstruct hierarchical relationships, complete the following steps:

1. Order item creation according to the hierarchy. Because hierarchical parents, hierarchical children, regular folder parents, and regular folder children are identified by their PIDs in the XML code, the order of item creation is

important. The hierarchy must be re-created from the top down, with parent folders being created before their children.

2. Manually edit the exported XML for hierarchical items.
   a. After the parent folder for a hierarchical item has been created, edit the exported XML for the item to add a <cm:parentFolder> element to the <cm:properties> elements and specify the PID of the new hierarchical parent folder.
   b. If other folder relationships with invalid PIDs appear in the <cm:folderSources> or <cm:folderContents> elements, remove them from the XML code before you create the item. This removal prevents any automated migration of items with their hierarchical relationships, ensuring that you reconstruct the hierarchy manually.

Mapping a user-defined schema to a storage schema with the XML schema mapping tool

IBM 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 an 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 to 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 reopen a previously created mapping (by 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 by 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 by 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 by 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 these APIs, which include classes `DKSchemaConverter`, `DKDocumentConverter`, and `DKMapperException` (or any `com.ibm.mm.sdk.xml.schema.*` classes) on z/OS for the schema transformation. However, the transformation can be performed by using the XML schema mapping tool on a non-z/OS platform (such as Windows) first and the resultant XSD or XML file can then be imported into IBM 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 IBM Content Manager server on z/OS).
- The transformed XSD/XML file can be downloaded onto z/OS and imported by 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 IBM 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 reference

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 IBM Content Manager connector XML APIs and the IBM 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 IBM 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 might take one of two forms. The PID of the item as it is stored in IBM 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 icmnlsdb13 XYZ_InsPolicy59 26 A1001001A04I10B45558I8302118
93 A04I10B45558I830211 14 1029&server=icmnlsdb&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 user ID = 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,
  user ID, password, entity, condition);
System.out.println("Search reply: " + reply.getAsString());
   }
static public CMBXMLMessage search(CMBXMLServices xmlServices,
String dstype, String server, String user ID, String password,
String entity, String condition)
  throws CMBException, Exception
      {
          return search(xmlServices, dstype, server, user ID, password,
    entity, condition, null);
      }
static public CMBXMLMessage search(CMBXMLServices xmlServices,
String dstype, String server, String user ID, String password,
String entity, String condition, String maxResults)
  throws CMBException, Exception
  {
  // Create the query string
  int     queryType = CMBBaseConstant.CMB_QS_TYPE_XPATH;
  String  queryString = "/" + entity;
  queryString += "[" + condition + "]";
  String connectString = "";
  // If the server name is followed by a parenthesized string,
  // use that string for the connect string.
  // e.g. ICMNLSDB(SCHEMA=ICMADMIN)
  if (server.indexOf("(") > 0) {
    connectString = server.substring(server.indexOf("(") + 1);
    server = server.substring(0, server.indexOf("("));
    if (connectString.endsWith(")")) {
      connectString = connectString.substring(0,
      connectString.length() - 1);
    }
  }
// continued...
```

Then to send an XML search request by using the above example, you can pass in your *server name* (`ICMNLSDB` 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>" + user ID + "</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 by 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 IBM Content Manager servers with ListServerRequest" on page 499

"Authenticating Web service requests for security by using XML" on page 500

"Changing a password with XML requests" on page 501

"Listing IBM Content Manager entities with ListSchemaRequest" on page 501

"Creating IBM Content Manager items with CreateItemRequest" on page 503

"Searching IBM Content Manager items with RunQueryRequest" on page 504

"Retrieving IBM Content Manager items with RetrieveItemRequest" on page 507

"Viewing your user privileges with XML requests" on page 512

“Working with IBM Content Manager folders through XML requests” on page 513

“Updating IBM Content Manager items with an XML UpdateItemRequest” on page 515

“Deleting IBM Content Manager items with DeleteItemRequest” on page 522

“Checking IBM Content Manager items out and in with CheckoutItemRequest and CheckinitemRequest” on page 523

“Linking IBM Content Manager items with CreateLinkRequest or DeleteLinkRequest” on page 524

“Moving IBM Content Manager items between entities with MoveItemRequest” on page 526

“Accessing IBM Content Manager document routing by using XML-based requests” on page 527

“Batching multiple requests in XML requests” on page 543

Listing IBM Content Manager servers with ListServerRequest

Create a <ListServerRequest> element to list all available IBM Content Manager servers.

To list all available IBM Content Manager servers, create a <ListServerRequest> element 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> element, then the request returns definitions of all IBM 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, IBM Content Manager returns an XML <ListServerReply> element 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>icmnlsdb</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 by using XML

Every time that you make a request to the Web services, you must pass in an IBM Content Manager user ID and password, or a WebSphere credential token associated with an IBM 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 view privileges only, 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, use SSL to send your SOAP requests.

To authenticate your Web service requests, create an <AuthenticationData> element. 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=`*`user ID`* where *user ID* is the schema of the IBM 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 IBM Content Manager servers with ListServerRequest

Changing a password with XML requests

<ChangePasswordRequest> changes your password in IBM Content Manager.

To change your password in IBM 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>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>testuser</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<NewPassword>passw0rd2</NewPassword>
</ChangePasswordRequest>
```

As a result, IBM 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 IBM Content Manager entities with ListSchemaRequest

<ListSchemaRequest> helps list all entities where schemas are required.

To list all IBM 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>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<EntityList all="true"></EntityList>
</ListSchemaRequest>
```

As a result, IBM 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 IBM Content Manager items with CreateItemRequest

<CreateItemRequest> helps create items.

To create items in IBM 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 IBM 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_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>
</CreateItemRequest>
```

As a result, IBM 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&amp;server=icmnlsdb&amp;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 connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item><ItemXML><XYZ_ClaimForm XYZ_ClaimFName="Joannifer"
  XYZ_ClaimLName="Smith" XYZ_ClaimNumber="8-123456" XYZ_DriversLic=
  "B12004960" XYZ_InsrdFName="Nicholas" XYZ_InsrdLName=
  "Smith" XYZ_PolicyNum="57904965371" XYZ_IncDate="2001-01-20">
  <ICMBASE><resourceObject MIMEType="image/tiff"
  xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"><label
  name="claimForm" /></resourceObject></ICMBASE>
  <ICMBASE><resourceObject MIMEType="image/jpeg"
  xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"><label
  name="claimPhoto"/></resourceObject></ICMBASE>
  </XYZ_ClaimForm>
</ItemXML></Item>
</CreateItemRequest>
```

As a result, IBM 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/CMBGetPIDUrl?pid=93
  3 ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B33016C1746518
  A04I10B33016C174651 14 1027&amp;server=icmnlsdb&amp;dsType=ICM"/>
</CreateItemReply>
```

Related information

Authenticating Web service requests for security

Working with XML services (Java only)

Searching IBM Content Manager items with RunQueryRequest

<RunQueryRequest> searches for specific items.

To search for specific IBM 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 [ quertyType="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, IBM 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&amp;server=icmnlsdb&amp;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 IBM Content Manager items with RetrieveItemRequest

<RetrieveItemRequest> searched and retrieves items as Web links or attachments.

To search for specific IBM 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 IBM Content Manager server. Each item element specifies an item to retrieve on the IBM 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 IBM 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\cmbxmlservices.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, IBM 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&amp;server=icmnlsdb&amp;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/icmrm/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="1RIO35P5RU5435209"
        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/icmrm/ICMResourceManager?order=retrieve&amp;
             item-id=A1001001A07C01B20133C21222&amp;version=1&amp;
             collection=CBR.CLLCT001&amp;libname=icmnlsdb&amp;
             update-date=2007-03-01+20%3A01%3A33.650668&amp;
             token=A4E6.FT6Fo_6_WbFRyL48RAw;&amp;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, IBM 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&amp;server=icmnlsdb&amp;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_InsrdLName="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

The <GetPrivilegesRequest> element allows you to view privileges on a specific item.

To view your privileges on a specific IBM Content Manager item (including user privileges), create a <GetPrivilegesRequest> element that identifies the following XML schema information:

```
<GetPrivilegesRequest>
  <AuthenticationData> ... </AuthenticationData>
  <Item URI=string />
</GetPrivilegesRequest>
```

<GetPrivilegesRequest> elements

<AuthenticationData> (required)
: Identifies the content server (<ServerDef>) and 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> element or 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>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"/>
</GetPrivilegesRequest>
```

As a result, IBM 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; that is, 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 IBM Content Manager folders through XML requests

A *folder* contains zero or more IBM 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 by 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 IBM 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

```
<RetrieveFolderItemRequest [ 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 IBM 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 IBM 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, IBM 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&amp;server=icmnlsdb&amp;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, IBM 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, IBM 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 IBM Content Manager items with an XML UpdateItemRequest

Retrieving IBM Content Manager items with RetrieveItemRequest

Updating IBM Content Manager items with an XML UpdateItemRequest

<UpdateItemRequest> updates items.

To update items from IBM 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 IBM 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 connectString="SCHEMA=ICMADMIN" configString="">
  <ServerDef><ServerType>ICM</ServerType>
  <ServerName>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
  A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029" />
<Add>
  <ChildItem parentURI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
    A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029">
    <ItemXML><XYZ_Insured XYZ_InsrdFName="Alice"
    XYZ_InsrdLName="Smith" /></ItemXML></ChildItem>
</Add>
<Replace>
  <Attribute name="XYZ_Street"><Value>123 Cedar Rd</Value></Attribute>
  <Attribute name="XYZ_ZIPCode"><Value>90543</Value></Attribute>
  <ChildItem parentURI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
  A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029">
    <ItemXML><XYZ_VIN
      xmlns:cm="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema"
      XYZ_VIN="1RIO35P5RU5435209" 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, IBM 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&amp;server=icmnlsdb&amp;dsType=ICM"/>
</UpdateItemReply>
```

"Adding objects inside IBM Content Manager items with an XML UpdateItemRequest" on page 518

"Replacing objects inside IBM Content Manager items with an XML UpdateItemRequest" on page 519

"Deleting objects inside IBM Content Manager items with an XML UpdateItemRequest" on page 521

Related information

Authenticating Web service requests for security

Adding objects inside IBM 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 IBM 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, IBM 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&amp;server=icmnlsdb&amp;dsType=ICM"/>
</UpdateItemReply>
```

Related information

Working with XML services (Java only)

Replacing objects inside IBM 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 existingContent 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="1RIO35P5RU5435209" 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, IBM 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&amp;server=icmnlsdb&amp;dsType=ICM"/>
</UpdateItemReply>
```

Deleting objects inside IBM Content Manager items with an XML UpdateItemRequest

Using the <Delete> element in an <UpdateItemRequest>, you can delete the XML schema values from existing IBM Content Manager items (this removes only this item's references to the attribute values).

To delete the following XML schema values from items, see the following example:

```
<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>icmnlsdb</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>passw0rd</Password></LoginData>
</AuthenticationData>
<Item URI="93 3 ICM8 icmnlsdb13 XYZ_InsPolicy59 26
  A1001001A04I10B35710G9498818 A04I10B35710G949881 14 1029" />
<Delete>
  <ChildItem URI="91 3 ICM8 icmnlsdb11 XYZ_Insured59 26
    A1001001A04I10B35710G9498818 A04I10B35710J374681 14 1030" />
</Delete>
</UpdateItemRequest>
```

As a result, IBM 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&amp;server=icmnlsdb&amp;dsType=ICM"/>
</UpdateItemReply>
```

Deleting IBM Content Manager items with DeleteItemRequest

<DeleteItemRequest> deletes items.

To delete items from IBM 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 IBM 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 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" />
</DeleteItemRequest>
```

As a result, IBM 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 IBM Content Manager items out and in with CheckoutItemRequest and CheckinitemRequest

When you check out items from IBM 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 IBM 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 IBM 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>ecl1ent</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, IBM 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 IBM 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 IBM 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>ecl1ent</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, IBM 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 IBM Content Manager items with CreateLinkRequest or DeleteLinkRequest

<CreateLinkRequest> or a <DeleteLinkRequest> creates or deletes items.

To create or delete links between items in IBM 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 IBM 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 A1001001A04I10B44211H0825818
  A04I10B44211H082581 14 1029&server=icmnlsdb&dsType=ICM">
<links xmlns="http://www.ibm.com/xmlns/db2/cm/api/1.0/schema">
  <outbound toItem=
  "http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93 3 ICM8
  icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B44213F1658218
  A04I10B44213F165821 14 1027&server=icmnlsdb&dsType=ICM"
  linkType="Contains"/></links>
</Item>
</CreateLinkRequest>
```

As a result, IBM 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, IBM 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 463

Related information

Authenticating Web service requests for security

Working with XML services (Java only)

Moving IBM Content Manager items between entities with MoveItemRequest

To move an IBM Content Manager item from one entity to another, create a <MoveItemRequest> element.

The attribute values in brackets in the <MoveItemRequest> element are optional. This move modifies the item tree of the attribute and child values, not the document content:

```
<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 reindex.

<NewValues newEntityName=*string*> (required)
: Specifies the new value to set in the entity. The value must be an XML specification of the item tree, conforming to the IBM Content Manager data model.

<MoveItemRequest> attributes

checkout (optional)
: Toggles whether to check out the item (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 reindexes (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, IBM 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 icmnlsdb11 CLAIM2_104759 26
  A1001001A04I13B04803F2085118 A04I16B41357H241001 14 1016"/>
</MoveItemReply>
```

Related information

Authenticating Web service requests for security

Accessing IBM Content Manager document routing by using XML-based requests

IBM Content Manager provides certain Web services that can help route your documents though a business process by using the APIs.

Important: To define document routing processes in IBM 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"

"Listing document processes with XML requests" on page 531

"Listing worklists with XML requests" on page 532

"Listing work packages with XML requests" on page 533

"Listing actions in IBM Content Manager with XML requests" on page 536

"Updating a work package with UpdateWorkPackageRequest" on page 537

"Starting a document routing process with StartProcessRequest" on page 538

"Ending a process with TerminateProcessRequest" on page 539

"Continuing a process with ContinueProcessRequest" on page 540

"Suspending a process with SuspendProcessRequest" on page 541

"Resuming a process with ResumeProcessRequest" on page 543

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 IBM Content Manager work nodes, create a <ListWorkNodesRequest> element that contains the following information:

```
<ListWorkNodesRequest>
  <AuthenticationData> ... </AuthenticationData>
</ListWorkNodesRequest>
```

As a result, IBM 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 routine 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 valid only 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 routine 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 routine 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 IBM 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 want a list of only the process names.

As a result, IBM 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 IBM 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 want a list of the worklist names only.

As a result, IBM 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 IBM 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.

<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, IBM Content Manager returns the work package in an XML <ListWorkPackagesReply>, <ListNextWorkPackagesReply>, or <ListWorkPackageCheckedOutOptionRequest> by using the following format:

```
<WorkPackage>
  <WorkPackageURI>aniURI</WorkPackageURI>
  <ItemIDURI>string</ItemIDURI>
  <ProcessName>string</ProcessName>
  <WorkNodeName>string</WorkNodeName>
[ <WorkPackageOwner>string</WorkPackageOwner> ]
[ <Priority>nonNegativeInteger</Priority> ]
[ <SuspendState>false</SuspendState> ]
[ <NotifyState>false</NotifyState> ]
[ <NotifyTime>yyyy-mo-dd-hh.mi.ss</NotifyTime> ]
[ <ResumeList><ResumeListDefinition>
    <RequiredItemTypeName>string</RequiredItemTypeName>
      <QuantityNeeded>0</QuantityNeeded>
    </ResumeListDefinition> ]
  </ResumeList> ]
[ <ResumeTime>yyyy-mo-dd-hh.mi.ss</ResumeTime> ]
[ <TimeLastMoved>yyyy-mo-dd-hh.mi.ss</TimeLastMoved> ]
[ <UserLastMoved>string</UserLastMoved> ]
[ <ProcessCompletionTime>yyyy-mo-dd-hh.mi.ss</ProcessCompletionTime> ]
[ <ContainerData><NameValuePair>
    <Name>attribute</Name><Value>string</Value>
  </NameValuePair>
[ <NameValuePair> ... </NameValuePair> ]
  </ContainerData> ]
</WorkPackage>
```

<WorkPackage> elements

<WorkPackageURI> (required)
Indicates the persistent identifier of the work package.

<ItemIDURI> (required)
Indicates the persistent identifier of the item that the work package is associated with.

<ProcessName> (required)
Indicates the name of the process that the work package is in.

<WorkNodeName> (required)
Indicates the name of the work node that the work package is currently located at.

<WorkPackageOwner> (optional)
Indicates the owner of the work package.

<Priority> (optional)
Indicates a user-defined, arbitrary priority for the work package. For example, 1, 2, 3...

<SuspendState> (optional)
Indicates whether the work package has been suspended in a process.

<NotifyState> (optional)
Indicates whether the work package notifies users when it moves to another work node.

<NotifyTime> (optional)
Indicates the time (in military format: *yyyy-mo-dd-hh.mi.ss*) when the work package was moved into notify state.

<ResumeList> (optional)
Only applies to a work package folder, and requires a <ResumeListDefinition> tag. Keeps the process suspended until a certain item type (specified in <RequiredItemTypeName>) of a specific quantity (specified in <QuantityNeeded>) appears in the folder. The default 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 IBM 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 IBM 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, IBM 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 IBM 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 routine to call when the user selects this action. For example, `C:\myapp.jsp`. The exit routine 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 IBM 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>cmdb2ls</ServerName></ServerDef>
  <LoginData><UserID>icmadmin</UserID><Password>o8rmond</Password>
  </LoginData>
</AuthenticationData>
<WorkPackageURI>89 3 ICM7 cmdb2ls11 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, IBM Content Manager returns an XML <UpdateWorkPackageReply> that indicates success.

Starting a document routing process with StartProcessRequest

The <StartProcessRequest> element starts a document routing process.

To start an IBM Content Manager through a document routing process, create a <StartProcessRequest> element that identifies the following information (elements in brackets are optional):

```
<StartProcessRequest>
  <AuthenticationData> ... </AuthenticationData>
  <ProcessName>string</ProcessName>
[ <WorkPackageOwner>string<WorkPackageOwner> ]
[ <Priority>nonNegativeInteger</Priority> ]
  <ItemIDURI> ... </ItemIDURL>
[ <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 IBM 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>cmdb2ls</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 cmdb2ls8 Claims2859 26
  A1001001A04H17B32800G0799718 A04H17B32800G079971
  14 1007</ItemIDURI>
</StartProcessRequest>
```

As a result, IBM Content Manager returns an XML <StartProcessReply> element, which 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>icmnlsdb</ServerName></ServerDef>
  <LoginData><UserID>icmadmin</UserID>
  <Password>aPassword</Password></LoginData>
</AuthenticationData>
<WorkPackageURI>89 3 ICM7 cmdb2ls11 WORKPACKAGE58 26
A1001001A04H17B32803I2340818 A04H17B33216H440631 03
204</WorkPackageURI>
</TerminateProcessRequest>
```

As a result, IBM 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

The <ContinueProcessRequest> element continues the work package (identified by the URI) to the next work node. This element 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 an IBM Content Manager item through the document routing process, create a <ContinueProcessRequest> element 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 can 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>cmdb2ls</ServerName></ServerDef>
  <LoginData><UserID>icmadmin</UserID>
  <Password>o8rmond</Password></LoginData>
</AuthenticationData>
<WorkPackageURI>89 3 ICM7 cmdb2ls11 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, IBM Content Manager returns an XML <ContinueProcessReply> element, which 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> element. This element checks out the item in the work package, and keeps the SuspendState attribute of the work package set to true for a specific duration. For work package folders, you can define multiple <ResumeListDefinition> elements that suspend the process until certain item types and quantities arrive.

To suspend an IBM Content Manager document routing process, create a <SuspendProcessRequest> element 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)
This element applies only to a work package folder. It 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 five 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>cmdb2ls</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>o8rmond</Password></LoginData>
</AuthenticationData>
<WorkPackageURI>89 3 ICM7 cmdb2ls11 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, IBM Content Manager returns an XML `SuspendProcessReply` element, which 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 IBM 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>cmdb2ls</ServerName></ServerDef><LoginData>
  <UserID>icmadmin</UserID><Password>o8rmond</Password></LoginData>
</AuthenticationData>
<WorkPackageURI>89 3 ICM7 cmdb2ls11 WORKPACKAGE58 26
  A1001001A04H17B32803I2340818 A04H17B33118B569521 03
  204</WorkPackageURI>
</ResumeProcessRequest>
```

As a result, IBM Content Manager returns an XML <ResumeProcessReply> that indicates success.

Related information

Authenticating Web service requests for security

Batching multiple requests in XML requests

You can combine multiple requests in a single message by 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 by 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 A1001001A04I10B44211H0825818
  A04I10B44211H082581 14 1029&server=icmnlsdb&dsType=ICM" />
</DeleteItemRequest>

<DeleteItemRequest
  xmlns="http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema">
  <AuthenticationData connectString="SCHEMA=ICMADMIN" configString="">
  ...
  </AuthenticationData>
  <Item URI="http://hostname/CMBGenericWebService/CMBGetPIDUrl?pid=93
  3 ICM8 icmnlsdb13 XYZ_ClaimForm59 26 A1001001A04I10B44213F1658218
  A04I10B44213F165821 14 1027&server=icmnlsdb&dsType=ICM" />
</DeleteItemRequest>
```

```
</Requests>
</Transaction>
</BatchRequest>
```

As a result, IBM 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 an IBM 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 IBM 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 IBM 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).

"IBM Content Manager Web services"

IBM Content Manager Web services

A *Web service* interface is a reusable, loosely coupled, software component that can be located, published, and started through a network, such as the Web.

IBM 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 an IBM Content Manager system.

With the Web services interface, you can dynamically integrate your applications with IBM Content Manager, regardless of the programming language in which they were written and the operating system on which they run. You can use the Web services interface to perform a task as simple as viewing 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 IBM Content Manager Version 8.4

IBM Content Manager Version 8.4 Web services has the following features:

- Supports only WebSphere Application Server versions 6.1 and higher. WebSphere Application Server versions 5.1.1 and 6.0.x are no longer supported.
- 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 a Web services application.
- Supports updated Websphere connection pools.

Prior to installation, you must configure your WebSphere Web server for Web services applications. In IBM Content Manager version 8.4, the Web services configuration program does not create a dedicated Web server to support Web services. Instead, Web services must be installed on an existing Web server.

IBM Content Manager Web services implementation

The IBM 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 that uses 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.

"IBM Content Manager Web services architecture" on page 550

"Comparison of the IBM Content Manager Web services interfaces" on page 551

"Web services functions" on page 552

"List of Web services requests" on page 553

"Hierarchical item operations using Web services" on page 556

IBM Content Manager Web services architecture

The main components in the IBM Content Manager Web services architecture include the requester, the Web services server, the XML beans layer, and the IBM 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 IBM 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 IBM Content Manager XML beans layer.
4. The XML beans transform the XML into multiple calls to the underlying IBM 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 IBM Content Manager Web services server.
8. The Web services server creates a SOAP message.
9. The SOAP message is returned to the requester.

Figure 28 depicts the steps involved in document processing by 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.

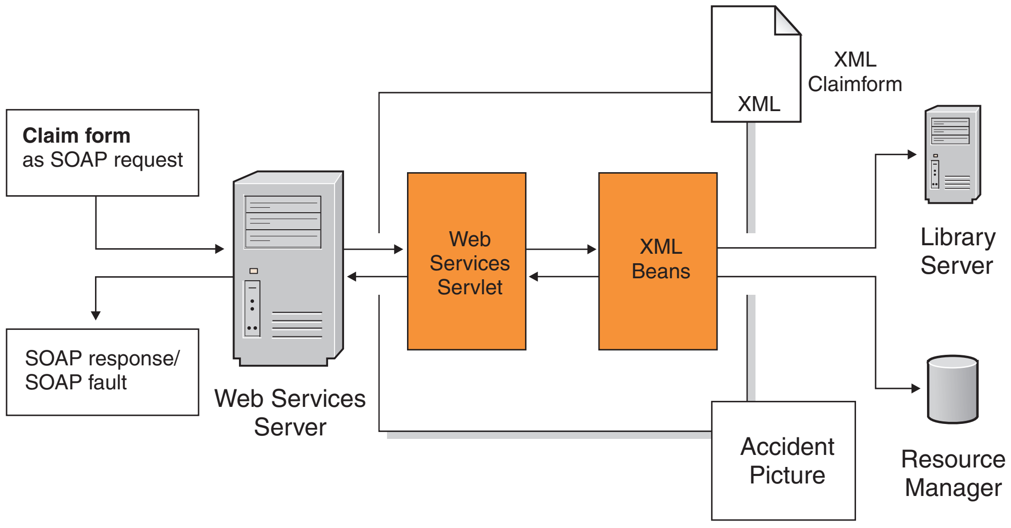


Figure 28. Processing a document by using Web services

Comparison of the IBM Content Manager Web services interfaces

IBM Content Manager offers the same Web services functionality through two different interfaces: `CMBGenericWebService` and `CMWebService` according to the different requirements for integrating IBM Content Manager Web services into customer applications.

The CMWebService interface defines all its operations in detail in the WSDL. The WSDL references 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 strongly typed, which prevents the errors that can occur when assembling or disassembling the XML message manually. These proxy classes are referred to as the generator style of a Web services interface.

Optionally, you can use XML schemas for specified IBM Content Manager item types to generate client proxy classes. These client proxy class APIs are detailed so that the client application developers do not need to know the syntax of the XML messages. Developers can follow the object-oriented programming guidelines to use these proxy classes to assemble request XML messages and disassemble response XML messages, which facilitates client application development. However, if the item types are changed, added, or removed from the definition, the developers must obtain the new XML schema for the item types, regenerate the proxy classes, and recompile the application with the new classes.

If the XML schema for the specific item types are not included when generating the proxy classes with the 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. By not including the specific item types in the proxy classes, client applications have the flexibility to work with any item types.

The WSDL of the CMBGenericWebServices interface 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 IBM 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 grants the IBM Content Manager Web services server the ability to add, change, or remove the Web services operations and their parameters in the future, but does not force the client application developers to use the new WSDL and recompile their application.

However, because of this flexibility of the CMBGenericWebServices interface, client application developers must have full knowledge of the Web services operations and the request and response XML messages. Developers must assemble or disassemble the message manually. Developers can use the following references: the CMBGenericWebServices WSDL, 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 in which IBM Information

Integrator for Content is installed), and the Web services samples that are included with IBM 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, consider the SOAP attachment capacity when choosing the Web services endpoint.

The attachment types supported by CMWebServices are:
- Message Transmission Optimization Mechanism (MTOM)
- Direct Internet Message Encapsulation (DIME)
- Multipurpose Internet Mail Extensions (MIME)

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 such details as building SOAP messages to be hidden so that the client only needs to build and send requests to a local proxy version of the Web service.

IBM Content Manager Web services server responds to any request messages that comply with the interfaces. Response messages are also based on the interfaces. 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. The following list shows which client development tools are supported by each attachment type:
- MTOM: Microsoft .NET Framework SDK version 2.0 or 3.0 and Web Services Enhancements (WSE) SDK version 3.0
- DIME: Microsoft .NET Framework SDK version 1.1 and Web Services Enhancements (WSE) SDK version 1.0 sp1
- MIME: IBM WebSphere client tools

Among non-IBM Web services software, Apache Axis supports both DIME and MIME and 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 Information Integrator for Content.

Related concepts

"Web services samples" on page 586

Related tasks

"Generating the WSDL file for the CMWebService Web service from system administration client" on page 568

Web services functions

The Web services do not support the full set of IBM Content Manager operations.

The functions supported by the Web services include the most commonly used run time 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"

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 586

Hierarchical item operations using Web services

Content Manager EE Web services supports hierarchical item operations.

If an input is provided in the XML code that violates the constraints of the hierarchical folder feature (for example, a nonunique ICM$NAME name under the parent), the request fails with the error code specific to the constraint that was violated. Content Manager EE Web services supports the following hierarchical item operations:

Creating a hierarchical item
: Use the <CreateItemRequest> element to create a hierarchical item. The XML code allows you to specify the ICM$NAME attribute value and the hierarchical parent folder DDO.

Retrieving a hierarchical item name
: Use the <RetrieveItemRequest> and <RunQueryRequest> elements to display the value of the ICM$NAME attribute if attributes are requested to be retrieved.

Reindexing a hierarchical item
: During a reindex, Web services allows you to change the item type and the ICM$NAME value of the item, but not the parent folder of the hierarchical item. You can reindex from non-hierarchical to hierarchical item types only if the system default folder is enabled.

Updating a hierarchical item name
: Use the <UpdateItemRequest> and <UpdateItemXMLRequest> elements to update a hierarchical item name if you set the new ICM$NAME value in the XML attributes.

Changing a hierarchical parent folder
: This operation is not supported by Web services.

Retrieving a hierarchical parent
: Use the <RetrieveFoldersForItemRequest> element to display all folder items referencing the hierarchical item, including its hierarchical parent. If all the item types are hierarchical, then only the parent is returned. If there

are multiple non-hierarchical folders containing the hierarchical item, it might not be obvious which one is the hierarchical parent.

Retrieving a hierarchical child item
Use the <RetrieveItemRequest> element to display all outbound links from `DKLinkCollection`, including any hierarchical child items that appear in the links collection, if the retrieve options specify link retrieval. The child items appear in the <cm:links> element. Hierarchical child items that appear in the `DKFolder` collection in the DDO are not displayed in the XML code of the item. They are not displayed because Web services does not pass in the flag to the XML API that specifies the embedding `DKFolder` information.

  RetrieveFolderItemsRequest displays all `DKFolder` child items of the hierarchical item. If all item types are hierarchical, then all the items returned are hierarchical child items. If not all of the items are hierarchical, hierarchical child items can be easily recognized because they contain the ICM$NAME attribute (as long as the retrieve options specify attribute retrieval).

Retrieving the hierarchical root folder
Use the <RunQueryRequest> element to return the root folder of the hierarchy by running a query on the Item ID, which is fixed, of the root folder.

Related information

Working with hierarchical item types

Installing and configuring the IBM Content Manager Web services

The following topics guide you through the installation and configuration of IBM Content Manager Version 8.4 Web services.

“Platforms supported by the Web services”

“Installation and configuration of Web services” on page 558

“Supported WebSphere Application Server topologies” on page 559

“Verifying, starting, and stopping the Web services” on page 560

“Post installation configuration” on page 561

“Logging and tracing” on page 563

“Un-installing and removing configuration” on page 563

Platforms supported by the Web services

You can use the Web services on various platforms.

Software prerequisites for the IBM Content Manager Web services server

WebSphere Application Server Version 6.1 Base and Network Deployment

Important: IBM 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 an IBM 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 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

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 IBM Content Manager Version 8.3 to IBM Content Manager Version 8.4.

In IBM Content Manager Version 8.3, the installation and configuration was done by using a single user interface (GUI). For example:

1. IBM 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.

IBM 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 by using two seperate GUIs provides flexibility because it is one of components in IBM Information Integrator for Content product in version 8.4.

Web services depends on ICM connector, which is a component of IBM Information Integrator for Content. Therefore, ICM connector must be installed if you choose to install Web services when you are installing IBM Information Integrator for Content. ICM connector must also be configured before configuring Web services.

Upgrading Web services from Version 8.3 to 8.4

IBM 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 IBM Content Manager Web services.

Deployment target

Web services is one of the components of IBM Information Integrator for Content. In IBM 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 IBM Information Integrator for Content is installed, which means that the deployment target (application server node) must be local to IBM Information Integrator for Content. Therefore, if IBM 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, IBM 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 IBM 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. IBM 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 IBM Information Integrator for Content is installed.

Supported topologies

The following topologies are supported by IBM 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 IBM 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 IBM Information Integrator for Content is installed. If it is local, it has the same topology as in unmanaged node. If it is remote, the IBM 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 IBM 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 IBM Information Integrator for Content is installed. The machine that has IBM 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 IBM Content Manager Version 8.3 to Version 8.4 does not support cluster. For more information, see Upgrading Web services to Version 8.4.

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 theWebSphere 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"

"Updating configuration files" on page 562

"Configuring WebSphere Application Server connection pooling for Web services" on page 562

Configuring shared library for JDBC driver

IBM Content Manager Version 8.4 Web services provides different solutions for different types of JDBC driver configurations.

IBM Content Manager Web services calls ICM connector APIs. The ICM connector API depends on JDBC driver to connect to library server databases. For IBM Content Manager Version 8.4 Web Services for multiplatforms with DB2, the JDBC driver is shipped with IBM 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 IBM 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 IBM 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.

IBM Content Manager Web services for Multiplatforms with Oracle, the Oracle JDBC driver JAR files are not packaged in the Web services EAR file. Instead, they must have already been installed on the workstation on that the Web services application is deployed. Web services configuration program creates the following shared library after deploying the application:

```
sharedLibrary_1_name=Oracle JDBC
sharedLibrary_1_classpath=${ORACLE_JDBC_DRIVER_PATH}/ojdbc14.jar
sharedLibrary_1_nativepath=${ORACLE_JDBC_DRIVER_PATH}
```

The shared libraries use an existing WebSphere variable in their paths ORACLE_JDBC_DRIVER_PATH. The administrator of the IBM Content Manager V8.4 Web services application must ensure that the value of the variable points to the directory where Oracle JDBC driver is 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 IBM Content Manager connector that are in the directory specified by CMCFGDIR property in file IBMCMROOT/cmgmt/cmbcmenv.properties

where *IBMCMROOT* is the installation directory of IBM 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 run time environment in the application server.

To synchronize these copies with the content of the originals, you must use theIBM 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.

Related information

Configuring Websphere Application Server connection pooling for use with the eClient

Logging and tracing

In IBM Content Manager Version 8.4, the Web services application has its own log file on each application server node.

IBM 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 IBM Information Integrator for Content. You can remove Web services configuration by using the configuration application for IBM 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 installation directory for IBM 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 IBM Content Manager Version 8 Release 4 Web services interfaces:
- CMWebService
- CMBGenericWebService

"Integrating CMWebService"
"Integrating CMBGenericWebService" on page 574
"Authenticating Web services requests for security" on page 578
"Creating a new instance of an item through Web services" on page 579
"Configuring single sign-on" on page 583

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 IBM 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`.

“Generating the WSDL file for the CMWebService Web service”
“Getting started with the CMWebService in a .NET environment” on page 569
“Generating proxy classes for .NET 1.1 client” on page 570
“Generating proxy classes for .NET 2.0 client” on page 571
“Programming Web services requests in a .NET environment” on page 572
“Consuming Web services over HTTPS (SSL)” on page 573

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 IBM 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 IBM 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 IBM 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 IBM 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,itemtype3b=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 IBM Content Manager Version 8.3 Fix Pack 4 or earlier. Without any change, these client applications can work well with IBM Content Manager 8.4 Web services server. If you want to regenerate the proxy classes by using new WSDL and schema files downloaded from IBM 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 the client applications.

cmdatamodel.xsd
: http://hostname/CMBSpecificWebService/services/CMWebServices/cmdatamodel.xsd

itemtype.xsd
: http://hostname/CMBSpecificWebService/services/CMWebService/itemtype.xsd?itemtype=itemtype1,itemtype2,itemtype3server=ICMNLSDB&credential=true&b=true, where

itemtype(required)
: See the explanation to the URL for downloading the `cmbmessages.xsd` file. The Web services extracts the specified item types and save their XML schema declarations in the `itemtype.xsd` file.

server(required)
: ICM server from which the Web services extracts 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 by using new WSDL and schema files downloaded from IBM 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 the client applications.

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 `CMBWebService.wsdl`, `cmbmessages.xsd`, and `cmdatamodel.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"

"Generating the WSDL file for the CMWebService Web service from command line" on page 569

Generating the WSDL file for the CMWebService Web service from system administration client:

Exporting item types to WSDL is still supported in IBM 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 IBM 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 IBM Content Manager Version 8.3 Fix Pack 4 or earlier.

Related reference

"Using the forceItemNamespace property when developing Web services client applications" on page 587

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 . Uisng 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 by using CMWebService which were developed with IBM 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 by 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.

"Enabling IPv6 support for Web services client applications that use .NET Framework 1.1"

Related concepts

"Programming Web services requests in a .NET environment" on page 572

Related tasks

"Generating the WSDL file for the CMWebService Web service from system administration client" on page 568

"Generating proxy classes for .NET 1.1 client"

"Generating proxy classes for .NET 2.0 client" on page 571

Enabling IPv6 support for Web services client applications that use .NET Framework 1.1:

The IBM Content Manager Web services server can be deployed in the IPv6-enabled network environment. To enable IPv6 support for .NET Framework 1.1 Web services, you must modify the `machine.config` file manually

Location of machine.config file

The `machine.config` file is stored in the `%WINDIR%\Microsoft.NET\Framework` folder in the directory where Microsoft Windows is installed. By default, it is located in the following path: `C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\CONFIG`.

Modifying machine.config file

The value of the ipv6-enabled configuration switch can be configured in the `machine.config` file. This value is false by default. It must be set to true to enable IPv6 support for .NET Framework 1.1. The following example shows the `ipv6 enabled` value set to true:

```
<system.net>
 <settings>
  ....
  //The following entry enables IPv6 support in the System.Net classes.
  //IPv6 support is predicated on availability of an IPv6 WinSock provider,
  //use of Windows XP, and the switch below being set to "true."

  <ipv6 enabled="true"/>
 </settings>
....
</system.net>
```

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:
   a. Open the `CMWebService.cs` file in an editor. Add an `by using` namespace clause. `by using Microsoft.Web.Services;`
   b. 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 {
      ```
   c. Save the `CMWebService.cs` file.

Generating proxy classes for .NET 2.0 client

To generate proxy classes, you must first install the software.

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\WseWsdl3.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 Web services client applications" on page 587

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.

Create an instance of an item
: The `CMWebServiceClient.cs` sample creates an instance of an `XYZ_InsPolicy` item.

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);
```

Attach binary content (if applicable)
: You can use one of three standard binary message formats to attach content parts to a request.

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;
         }
        }
```

Related information

Authenticating Web services requests for security

Creating a new instance of an item through Web services

Programming run-time operations through the XML JavaBeans

Attaching binary content parts to items in Web services

Consuming Web services over HTTPS (SSL)

To exchange confidential data such as passwords with IBM Content Manager Web services, use SSL to protect packages that are transferred over the Internet.

Important: These instructions are applicable only for a Web services client proxy class written in C#.

"Consuming Web services over HTTPS (SSL) for .NET Framework 1.1"

"Consuming Web services over HTTPS (SSL) for .NET Framework 2.0"

Consuming Web services over HTTPS (SSL) for .NET Framework 1.1:

You can write a client proxy class that communicates with IBM Content Manager Web services through HTTPS for .NET Framework 1.1.

To write a client application that communicates with IBM Content Manager Web services through HTTPS, follow this procedure:

1. Add two namespaces to the C# proxy class that is automatically generated by the WSDL tool based on the WSDL exported by the IBM Content Manager system administration client:

   ```
   by using System.Net;
   by using System.Security.Cryptography.X509Certificates;
   ```

2. Add a new class that inherits from the `ICertificatePolicy` class to the proxy class. For example:

   ```
   public class TrustAllCertificatePolicy : ICertificatePolicy
   {
     public TrustAllCertificatePolicy() {}
     public bool CheckValidationResult(ServicePoint sP,
      X509Certificate cert,WebRequest wReq, int certProblem)
     {
      if (wReq.RequestUri.AbsoluteUri == "https://host:port/websvc_url")
       return true;
      else
       return false;
     }
   }
   ```

3. Add the following line to the constructor of the proxy class:

   ```
   ServicePointManager.CertificatePolicy = new TrustAllCertificatePolicy();
   ```

Consuming Web services over HTTPS (SSL) for .NET Framework 2.0:

You can write a client proxy class that communicates with IBM Content Manager Web services through HTTPS for .NET Framework 2.0

To write a client application that communicates with IBM Content Manager Web services through HTTPS, follow this procedure:

1. Add three namespaces to the C# proxy class that is automatically generated by WSDL tool based on the WSDL exported by the IBM Content Manager system administration client:
   ```
   by using System.Net;
   by using System.Security.Cryptography.X509Certificates;
   by using System.Net.Security;
   ```
2. Define the `ServicePointManager.ServerCertificateValidationCallback` callback method. This method is equivalent to the `ICertificatePolicy.CheckValidationResult` method. For example:
   ```
   public bool CheckValidationResult(object sender,
      X509Certificate cert, X509Chain chain, SslPolicyErrors errors)
   {
     return true;
   }
   ```
3. Add the following lines to the constructor of the proxy class:
   ```
   ServicePointManager.ServerCertificateValidationCallback = new
   System.Net.Security.RemoteCertificateValidationCallback(CheckValidationResult);
   ```

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” on page 575.

“Getting the WSDL file for the CMBGenericWebService Web service”

“Customizing Java applications to work with CMBGenericWebService” on page 575

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.

   "Programming Web services requests in a Java environment"

Related concepts

"Programming Web services requests in a Java environment"

Related tasks

"Getting the WSDL file for the CMBGenericWebService Web service" on page 574

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 an 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

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 by using CMBGenericWebService endpoint can upload or download binary data to CMBGenericWebService endpoint that uses 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 information

Programming run-time operations through the XML JavaBeans

Web services samples

Authenticating Web services requests for security

Every time you make a request to the Web services, you must pass in an IBM Content Manager user ID and password, or a WebSphere credential token associated with an IBM 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, 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 IBM Content Manager library server database name (for example, *cmdb2ls*), user ID represents your IBM Content Manager user ID (for example, `icmadmin`), and *password* represents your login password.

C# sample

```
public AuthenticationData setupAuthenticationData(
  string server, string user ID, string password)
{

  AuthenticationData authData = new AuthenticationData();
  ServerDef serverDef = new ServerDef();
  serverDef.ServerName = server;
  authData.ServerDef = serverDef;
  AuthenticationDataLoginData loginData =
       new AuthenticationDataLoginData();
  loginData.UserID = user ID;
  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 IBM Content Manager library server database name (for example, `cmdb2ls`), *username* represents your IBM Content Manager user ID (for example, `icmadmin`), 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()});
}
```

"Attaching binary content parts to items in Web services"

Related concepts

"Items" on page 61

Attaching binary content parts to items in Web services

The IBM 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 IBM 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 IBM 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 *!ENTITY!* 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 IBM 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 IBM 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 IBM 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 needs to supply the user ID and password only one time.

To set up single sign-on (SSO) for the Web services, you must configure SSO for both IBM Content Manager and WebSphere Application Server. Then, you can connect to IBM Content Manager Version 8 server through Web services by 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 following URL in a Web browser: http://*<hostname>*/CMBSpecificWebService/index.html.
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 `Content Manager Specific Web Service` is printed in the Web browser.

"LTPA tokens and single sign-on"

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 workstation. There are various ways to obtain the credential information, which is stored as a cookie on the client workstation, 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 authentiation was successful,
  * any subsequest 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 identity the user.

Example

```
protected static final String AUTHENTICATION_DATA_TEMPLATE =
        "<AuthenticationData  connectString=\"SCHEMA=ICMADMIN\"
        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", "icmnlsdb", 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

For best performance, follow the suggestions that are outlined in the Performance tuning for Content Manager related to WebSphere Application Server. 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 onWebSphere 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"

"Using the forceItemNamespace property when developing Web services client applications"

"Updating the Web services server configuration to refer to the 32-bit libraries when using a DB2 64-bit instance" on page 588

"A Web services query retrieves the latest version of an item by default" on page 589

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, select a value that is less than 10 for the cached statements.

Using the forceItemNamespace property when developing Web services client applications

The `forceItemNamespace` property allows a namespace to be set for item type schema information.

In IBM Content Manager Version 8.4, the `forceItemNamespace` property in the `cmbxmlservices.properties` file is enabled and set to http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema by default. Do not change the value of this property unless you read the backward compatibility issues for Web services client applications developed with IBM Content Manager Version 8.3 (CMWebService endpoint).

In IBM Content Manager Version 8.3, the `cmbxmlservices.properties` file does not include the `forceItemNamespace` property by default. However, you can include it by manually adding the line `forceItemNamespace=http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema` to the `cmbxmlservices.properties` file.

If you have developed a Web services client application with IBM Content Manager Version 8.3 (CMWebService endpoint), the application does not retrieve items from theIBM Content Manager Version 8.4 Web services server when the `forceItemNamespace` property is not present in the`cmbxmlservices.properties` file or is commented out.

To solve this problem, you can do one of the following:

- Comment out the line `forceItemNamespace=http://www.ibm.com/xmlns/db2/cm/beans/1.0/schema` in the `cmbxmlservices.properties` file and restart the IBM Content Manager Version 8.4 Web services server. You do not need to change your application.
- Download or export the `WSDL` and `XML` schema files for IBM Content Manager Version 8.4 (CMWebService endpoint) and regenerate the proxy classes from the `WSDL` and `XML` schema files. Although the code of other parts of your application does not need to be changed, you must compile and rebuild the entire application.

Important: .NET 2.0/WSE 3.0 clients can work with the Content Manager Web Services server only when the `forceItemNamespace` property is enabled. The `forceItemNamespace` property must be enabled when generating specific WSDL and schema files for .NET 2.0/WSE 3.0 clients.

Updating the Web services server configuration to refer to the 32-bit libraries when using a DB2 64-bit instance

The IBM Content Manager APIs and Web services require the 32-bit libraries.

Symptom

When you send requests through IBM Content Manager Web services using a DB2 64-bit instance, you receive the following error: `DGL0394A: Error in DriverManager.getConnection; No suitable driver`.

Possible cause

When installing IBM 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. For an AIX system, click the *LIBPATH* variable. For a Solaris system, click the *LD_LIBRARY_PATH* variable. This variable contains one or more paths to library locations in the file system.
   d. The DB2 UDB libraries are found in the `DB2instancehome/lib` file path, where *DBinstancehome* is `/home/db2inst1/sqllib` by default.
   e. Change the `lib` file path to `lib32`.
   f. Click **OK** to save the new value.
   g. Save the changes to the master configuration.
3. Restart the cmwebsvc server.

A Web services query retrieves the latest version of an item by default

If you do not specify the version of the items to retrieve in a Web services query, the most recent version is retrieved by default.

Symptoms

When executing a Web services query, if you do not set the version of an item to retrieve in the version attribute, the most recent version is retrieved by default.

Causes

If the version attribute is not specified, it is implicitly set to `LATEST_VERSION`. This setting is the default behavior of Web services.

Resolving the problem

If you need a specific version of an item to be retrieved in a query, explicitly set the version attribute.

Working with the Java document viewer toolkit

The Java 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 and 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 IBM Content Manager applications.

Java document viewer toolkit overview

The Java document viewer toolkit supports viewing, annotation, and manipulation of various document formats. These formats are primarily: TIFF, MO:DCA (containing IOCA, PTOCA), GIF, JPEG, PCX, bitmap, and text documents and Microsoft Office documents by using a bridge to Oracle Outside In Technology.

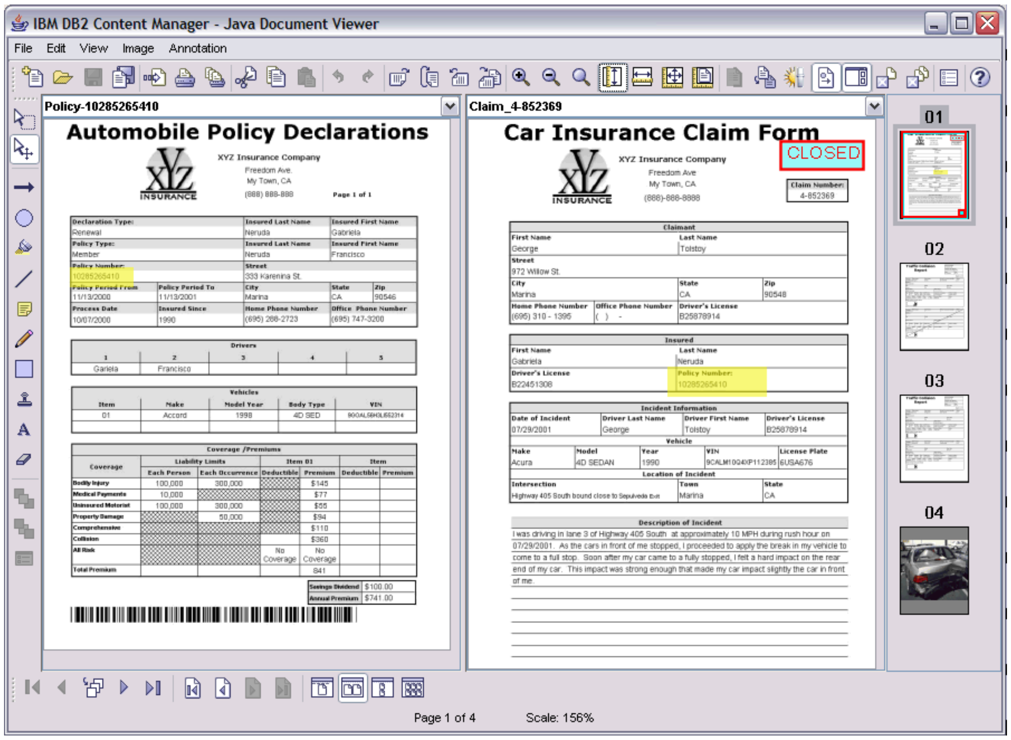
You can extend the viewer to support other document formats, by writing document engines that are classes that provide document format-specific functions.

The Java document viewer toolkit provides the following capabilities:

- Multiple-document interface with several different layouts including tab 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.
- When you are viewing two document views in side-by-side layout, you have the option of locking the scroll bars so that both document views can be scrolled simultaneously.
- Thumbnails view of the pages of a document: Clicking 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.
- Thumbnails can be configured either to be docked or to float in a separate window. Configuring thumbnails to be visible in a floating window allows you to view a document and the thumbnails simultaneously in separate windows.
- 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 document 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 the 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 box, circle, line, arrow, text, highlight, sticky note, pen, and stamp. Stamp annotations can be either text or images.
- Page manipulation, including copying, reordering, rotating, and deleting pages in image-paged document formats and the ability to save the changed document.
- Multiple selection of pages in the thumbnails, which is useful for rotation and other manipulations.
- 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 in 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.
- The ability to export a document, which can include annotations, to the file system.
- User controllable preferences, which are saved locally and are used in subsequent viewer sessions. These 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.
- The ability to use an optical character recognition engine. This capability allows the viewer to recognize and extract textual content from an image document so that its text can be searched and selected. The selected text can then be copied and pasted to other applications, such as Microsoft Word.

The following screen capture is an example of a generic Java document viewer that uses all of the default settings.



Automobile Policy Declarations

XYZ Insurance Company
Freedom Ave.
My Town, CA
(888) 888-888

Page 1 of 1

Declaration Type: Renewal	Insured Last Name Neruda	Insured First Name Gabriela
Policy Type: Member	Insured Last Name Neruda	Insured First Name Francisco
Policy Number: 10285265410	Street 333 Karenina St.	

Policy Period From	Policy Period To	City	State	Zip
11/13/2000	11/13/2001	Marina	CA	90546

Process Date	Insured Since	Home Phone Number	Office Phone Number
10/07/2000	1990	(695) 288-2723	(695) 747-3200

Drivers

1	2	3	4	5
Gariela	Francisco			

Vehicles

Item	Make	Model Year	Body Type	VIN
01	Accord	1998	4D SED	900ALS6H3L652314

Coverage /Premiums

Coverage	Liability Limits		Item 01		Item	
	Each Person	Each Occurrence	Deductible	Premium	Deductible	Premium
Bodily Injury	100,000	300,000	▨	$145		
Medical Payments	10,000	▨	▨	$77		
Uninsured Motorist	100,000	300,000	▨	$55		
Property Damage	▨	50,000	▨	$94		
Comprehensive	▨	▨	▨	$110		
Collision	▨	▨		$360		
All Risk	▨	▨	No Coverage	No Coverage		
Total Premium				841		

Savings Dividend	$100.00
Annual Premium	$741.00

Car Insurance Claim Form

CLOSED

XYZ Insurance Company
Freedom Ave
My Town, CA
(888)-888-8888

Claim Number: 4-852369

Claimant

First Name George	Last Name Tolstoy	
Street 972 Willow St.		
City Marina	State CA	Zip 90548
Home Phone Number (695) 310 - 1395	Office Phone Number ( ) -	Driver's License B25878914

Insured

First Name Gabriela	Last Name Neruda
Driver's License B22451308	Policy Number: 10285265410

Incident Information

Date of Incident	Driver Last Name	Driver First Name	Driver's License
07/29/2001	George	Tolstoy	B25878914

Vehicle

Make	Model	Year	VIN	License Plate
Acura	4D SEDAN	1990	9CALM10Q4XP112385	6USA676

Location of Incident

Intersection	Town	State
Highway 405 South bound close to Sepulveda Exit	Marina	CA

Description of Incident

I was driving in lane 3 of Highway 405 South at approximately 10 MPH during rush hour on 07/29/2001. As the cars in front of me stopped, I proceeded to apply the break in my vehicle to come to a full stop. Soon after my car came to a fully stopped, I felt a hard impact on the rear end of my car. This impact was strong enough that made my car impact slightly the car in front of me.

Figure 29. Generic Java document viewer

Related information

Working with page manipulation functions

Viewer architecture

The Java document viewer toolkit provides beans and classes to allow you to manipulate documents and annotations and control the document viewer interface.

The Java document viewer toolkit contains a Document Viewer bean (`CMBDocumentViewer`) and a Document Services bean (`CMBDocumentServices`), which provide a mechanism for integrating the Java document viewer into applications based on IBM Information Integrator for Content JavaBeans.

The Java document viewer toolkit also contains the Generic Doc Viewer (`CMBGenericDocViewer`), Streaming Doc Services (`CMBStreamingDocServices`), and Annotation Services (`CMBAnnotationServices`) classes. With these classes, you can use the Java document viewer in applications in which you cannot use the beans, such as in stand-alone viewing or in distributed process situations in which the connection to the content stores is not local.

The `CMBStreamingDocServices` class manages a set of document engines that parse documents, render pages, and provide you with the capability to manipulate the pages of a document. These document engines provide parsed document images in various formats, such as TIFF and IOCA, text and rich-text formats, and Microsoft Office formats. You can also write additional document engines and plug them into the Java document viewer architecture to support additional formats or alternative rendering for document formats.

With the `CMBAnnotationServices` class, you can manipulate annotations. The MSTech annotation engine, which the Java document viewer provides, parses only annotation formats that are specific to IBM Content Manager. However, you can write additional annotation engines to handle other annotation formats.

Starting in Version 8.4.2, the Java document viewer toolkit ships with viewer applet and servlet classes that your custom Web-based applications can use.

Starting in Version 8.4.3, the Java document viewer toolkit allows you to specify an optical character recognition engine.

The following diagram illustrates the components that make up the Java document viewer toolkit:

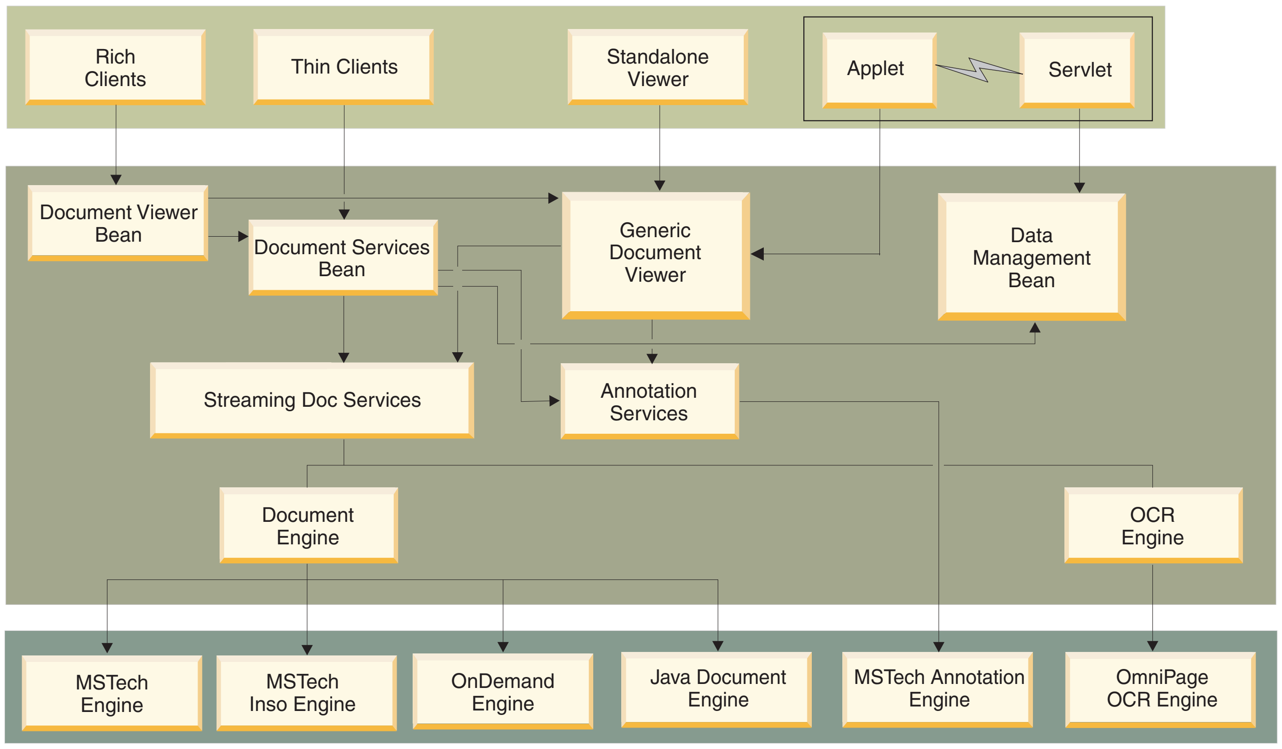


Figure 30. The components in the Java document viewer toolkit

"Document engines"

"Annotation engines" on page 595

"Optical character recognition engines" on page 595

"Example viewer architectures" on page 596

Document engines

The Java document viewer uses document engines to process documents. You can use a document engine to parse documents, render pages, and manipulate the pages of a document. A document engine can work with a set of document types, such as TIFF and JPEG.

The Java document 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 IBM 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 document engine (`CMBMSTechInsoEngine`)
This engine supports Microsoft Office, Lotus® SmartSuite®, and other Office document formats.

IBM Content Manager OnDemand document 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 URLs to HTML with a link to the URL.

Snowbound engine (CMBSnowboundEngine)
This engine supports PDF documents.

Although these document engines are public interfaces, do not program directly to them. Instead, use the interfaces provided by the `CMBDocumentServices` bean or the `CMBStreamingDocServices` class to access the functionality provided by the document engines.

Some of these document engines are not pure Java and have portability limitations, which can restrict use of the toolkit on some operating systems. The MS-Tech and Java document engines are pure Java, and can be used on most operating systems. The other document engines contain operating system-specific logic that restricts their use to Microsoft Windows only.

Related information

Supported document types and conversions in the Java document viewer toolkit

Annotation engines

The Java document viewer toolkit provides the MS-Tech annotation engine to retrieve and update IBM Content Manager annotations. You can also write your own annotation engine to handle other annotation formats.

The MS-Tech annotation engine supports IBM Content Manager Version 7 and later and VisualInfo for AS/400 annotations.

Optical character recognition engines

The Java document viewer toolkit defines a pluggable interface to enable support for an optical character recognition (OCR) engine. An OCR engine is designed to recognize text from an image, which helps you do tasks such as copy, paste, and search with screen readers.

A sample OCR engine implementation is provided for the Nuance OmniPage Capture Software Development Kit (SDK). If you have a license for the Nuance OmniPage Capture SDK, you can use this sample OCR engine to enable the Java document viewer for OCR technology. For more information, see the Java class file `com.ibm.mm.viewer.samples.ocr.OmnipageOCREngine.java` in the OCR samples. You can also write your own OCR engine by extending the Java class `CMBOCREngine`. For more information, see the *Application Programming Reference*.

When you use a valid OCR engine, the following tools are available: ocr_doc, find, and select_text. These tools can be added to a viewer toolbar in an engine properties file. For example:

```
OperationToolbar.tools=ocr_doc,select_text,find,separator,save_doc, ...
```

To specify an OCR engine in the Java document viewer toolkit, set the following properties to indicate the class name and path of your OCR engine. These properties can either be specified in an engine properties file, such as cmbviewerengine.properties, or specified programmatically:

- `OCRENGINE_CLASSNAME`
- `OCRENGINE_exePath`

The following example demonstrates how to specify an OCR engine in an engine properties file:

```
OCRENGINE_CLASSNAME=com.ibm.mm.viewer.samples.ocr.OmnipageOCREngine
OCRENGINE_exePath=c:\cm8\windows\nativelib
```

The following example demonstrates how to specify an OCR engine programmatically:

```
Properties engineCMProperties = new Properties();
engineCMProperties.put("ENGINES","2");
engineCMProperties.put("ENGINE1_CLASSNAME",
   "com.ibm.mm.viewer.mstech.CMBMSTechDocumentEngine");
engineCMProperties.put("ENGINE2_CLASSNAME",
   "com.ibm.mm.viewer.CMBSnowboundEngine");
engineCMProperties.put("OCRENGINE_CLASSNAME",
   "com.ibm.mm.viewer.samples.ocr.OmnipageOCREngine");
engineCMProperties.put("OCRENGINE_exePath",
   "c:\\cm8\\windows\\nativelib");

// Create document services.
docServices = new CMBStreamingDocServices(
   new StreamingDocServicesCallbacks(), engineCMProperties);
```

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"
"Java application" on page 597
"Thin client" on page 597
"Applet and servlet" on page 598
"Dual-mode with applet or servlet" on page 598

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.

The following figure illustrates a standalone viewer architecture.

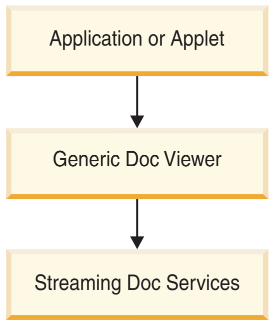


Figure 31. Standalone viewer

Java application

To create a production Java application use the IBM 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 open other viewers to view documents. Remember that these viewers might have operating system dependencies. The following figure illustrates the architecture that you can use to build a Java application.

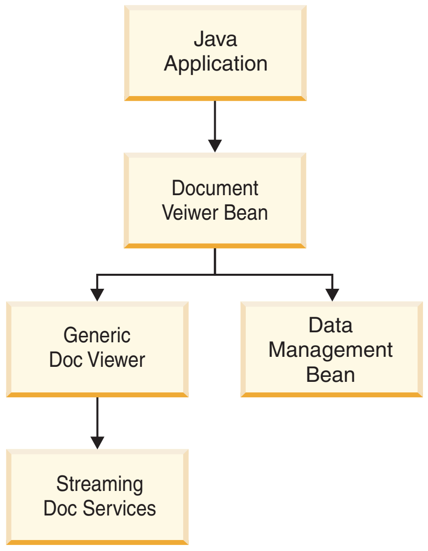


Figure 32. Java application

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 open) 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.

The following figure illustrates a thin client architecture.

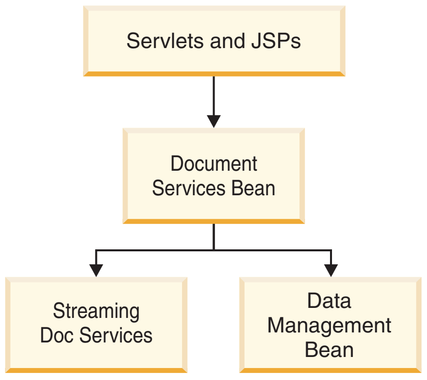


Figure 33. Thin client

Applet and servlet

A Web-based application can provide document viewing and annotation editing capabilities by using an applet and servlet approach. The eClient viewer applet uses this architecture.

The eClient viewer 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 obtains the requested information by using the `CMBDataManagement` bean. The following figure illustrates the applet and servlet architecture.

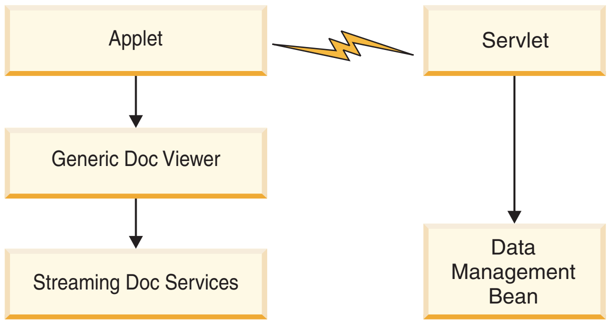


Figure 34. Applet and servlet

Starting in Version 8.4.2, the Java viewer toolkit has its own viewer applet (`CMBDocumentViewerApplet`) and servlet (`CMBDocumentViewerServlet`), which a Web-based application can use.

Related information

Invoking the viewer 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.

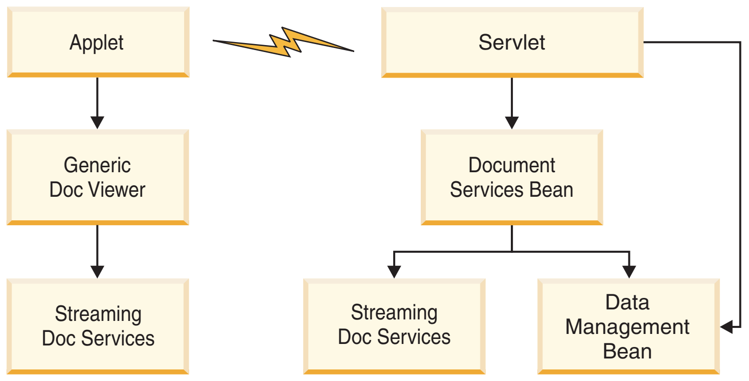


Figure 35. Dual-mode viewer

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”

“Working with documents and annotations” on page 601

“Customizing the viewer” on page 602

“Invoking the viewer applet and servlet” on page 608

Creating a standalone viewer application or applet

You can 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 by 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 by 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.properities 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 by using the add listener methods of CMBGenericDocViewer, such as addDocSaveListener.
10. Specify the engines and their order by 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.CMBSnowboundEngine");
    // 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 information

Customizing the viewer

Working with documents and annotations

To use the viewer toolkit components to work with documents and annotations, call the loadDocument method first to load the documents and loadAnnotationSet method to load 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 document viewer, provided by CMBGenericDocViewer, is included with the product. You can customize the functionality provided by the document 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 document viewer by providing a custom configProperties object while constructing the viewer. To view an example of a customized document viewer, see the TGenericDocViewer viewer application sample. The sample document 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.

The document viewer configuration contains default toolbars, such as **Operation**, **Annotation**, and **Page**, a number of default tools, and a few properties that control certain viewer behaviors such as page manipulation, multiple selection, and so on. By using a custom document 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 commas, 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, which the viewer implements. 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 O
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.

Defining an action class for a 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:

```
//submenus (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_actualsize
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

To automatically invert a document the first time it is opened, set the `Document.invert` property to true. Otherwise, to open a document normally, set the `Document.invert` property to false. By default, documents are not inverted. For 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. Enhancing certain image documents can increase memory usage. For example:

```
Document.enhance=true
```

Rotating documents

Documents can be set to rotate when you open them. The default behavior is to not rotate a document when it is opened. To automatically rotate a document when it is opened, set the `Document.rotate` property accordingly:

- Do not rotate: rotate_0
- Rotate 90 degrees: rotate_90
- Rotate 180: rotate_180
- Rotate 270: rotate_270

For example:

```
Document.rotate=rotate_0
```

Document scroll locking

When you view two document views in a side-by-side layout, you have the option of locking the scroll bars so that both document views can be simultaneously scrolled, therefore allowing you to visually compare them more easily. By default, document scroll locking is set to off when side-to-side document viewing is initiated.

Showing annotations

To automatically show all annotations associated with a document when it is opened, set the `Annotations.show` property to true. Set the property to false if you want the annotations hidden when the document is opened. The default setting is true. For example:

```
Annotations.show=true
```

Zooming

To zoom in and out on a document, set the `Zoom.factor` property to a numeric value greater than 0 and less than 100. This value represents a percentage. The default zoom value is 10 percent. For 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 by using the initial zoom (the default)

For 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 8.5 x 11-inch page. For example:

```
ThumbnailSize.width=60
ThumbnailSize.height=77
```

The thumbnail size might also be a factor of this width and height by using ThumbnailSize. For 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.widht * 2,
ThumbnailSize.height * 2
By default ThumbnailSize is medium
ThumbnailSize=medium
```

Thumbnail docking

Thumbnails can be configured either to be docked or to float in a separate window. Configuring thumbnails to appear in a floating window allows you to view a document and the thumbnails simultaneously in separate windows. By default, thumbnails are docked. To configure thumbnails to dock, set the Thumbnail.dock property to true. To configure thumbnails to float in a separate window, set the Thumbnail.dock property to false. For example:

```
Thumbnail.dock=false
```

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. For example:

```
PageManipulation=true
```

Customizing viewer behavior

The following 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 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
```

Invoking the viewer applet and servlet

Starting in Version 8.4.2, the Java viewer toolkit includes its own applet and servlet bean.

CMBDocumentViewerApplet

A viewer applet class that can view, annotate, edit, and manipulate documents through the Web. The viewer applet can be used with `CMBDocumentViewerServlet`.

The following example views a single-part document:

```
<applet width="100%" height="100%"
code="com.ibm.mm.viewer.applet.CMBDocumentViewerApplet"
archive="cmbview81.jar"> <param name="contentPartUrl1" value="http://
www.ibm.com/i/v14/t/ibm-logo.gif"> </applet>
```

You can pass the following parameters into the viewer applet:

`documentName`
: The name of the document. If this name is not specified, then the URL of the first part is used.

`contentPartCount`
: The number of parts (segments) that the document is composed of. The default is 1.

`contentPartUrl`*n*
: A URL to use to retrieve a content part, where *n* represents the number of the content part. For example, the first part equals 1.

`contentPartMimeType`*n*
: The MIME content type of the content part. If this parameter is not specified, then the MIME type defaults to the one returned from the request for the part.

: Use this parameter only to perform direct retrieval from the resource manager.

`annotationPartCount`
: The number of annotation parts (files, sets) that the document has. The default is 0.

`annotationPartUrl`*n*
: A URL to use to retrieve an annotation part, where *n* represents the number of the annotation part. For example, the first part equals 1.

: Although the annotationPartCount can be zero if no annotations exist, it is recommended that you specify 1 so that new annotations are saved.

`documentLockUrl`
: A URL to use to lock the document (if locking is supported). The viewer applet assumes that the file is not locked for a non-success response code.

`documenUnlockUrl`
: A URL to use to unlock the document (if locking is supported). The viewer applet automatically unlocks a locked document after it is edited and closed.

documentResourceUrlPrefix
A URL prefix (to which the resource name is appended) that retrieves shared document resources such as background images of pre-printed forms.

contentEditPrivilege
If set to true, the content can be edited (such as deleting and rotating pages). If set to false, then the content cannot be edited. The default is false.

Note that although you can temporarily rotate pages for non-editable content, the rotation is not saved when the document is closed.

annoationEditPrivilege
If set to true, the annotations can be edited. This setting includes creating new annotations and deleting existing annotations. The default is false.

exportPrivilege
If set to true, the document can be exported. The default is true.

printPrivilege
If set to true, the document can be printed. The default is true.

propertiesFile
A URL to the combined properties file for CMBGenericDocViewer, CMBStreamingDocServices, and CMBAnnotationServices. If not properties file is specified, then all defaults are used.

CMBDocumentViewerServlet

A document viewer servlet class that a Web application can use to view a document in IBM Content Manager (using the server, item ID, and log in credentials). The servlet returns HTML code that can start the viewer applet, `CMBDocumentViewerApplet`. The servlet can work as a stand-alone servlet or inside of another Web application.

The following example URL starts the servlet to view a document on an IBM Content Manager library server named *icmnlsdb*.

```
http://localhost:8080/TestWebApp/
CMBDocumentViewerServlet?serverName=icmnlsdb&amp;userid=icmadmin
&amp;password=password&amp;pid=<<pid>>
```

You can pass the following parameters into the servlet:

serverName
The name of the server. This parameter is required.

pid
The identifier for the item. This parameter is required.

serverType
The type of server, according to IBM Information Integrator for Content definitions. The default is ICM (the server type IBM Content Manager).

userid
The user ID to the server. If this parameter is not specified, then the servlet uses HTTP authorization to make the Web browser prompt for a user ID and password.

`password`
The password to the server. If this parameter is not specified, then the servlet uses HTTP authorization to make the Web browser prompt for a user ID and password.

`action`
The action for the viewer applet to perform for additional document resources or annotations. If no action is specified, then the servlet returns HTML code that starts the applet viewer only. The possible values are:

lock Locks a document.

unlock
Unlocks a document.

part Gets and puts a content part of a document.

resource
Gets a shared resource. Put is not supported. If you specify this parameter, then the resourceID parameter is required.

annotationPart
Gets and puts an annotation part of a document.

`partNum`
The number of the part (for actions that work with parts). The default is 1.

`useDirectUrls`
If set to true, then the servlet uses the URLs to the IBM Content Manager resource manager (rather than streaming content through this servlet). The default is false.

`resourceId`
The identifier for the shared resource (used with the getResource action).

Working with the annotation services

The IBM 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. The following figure shows the annotation services class diagram.

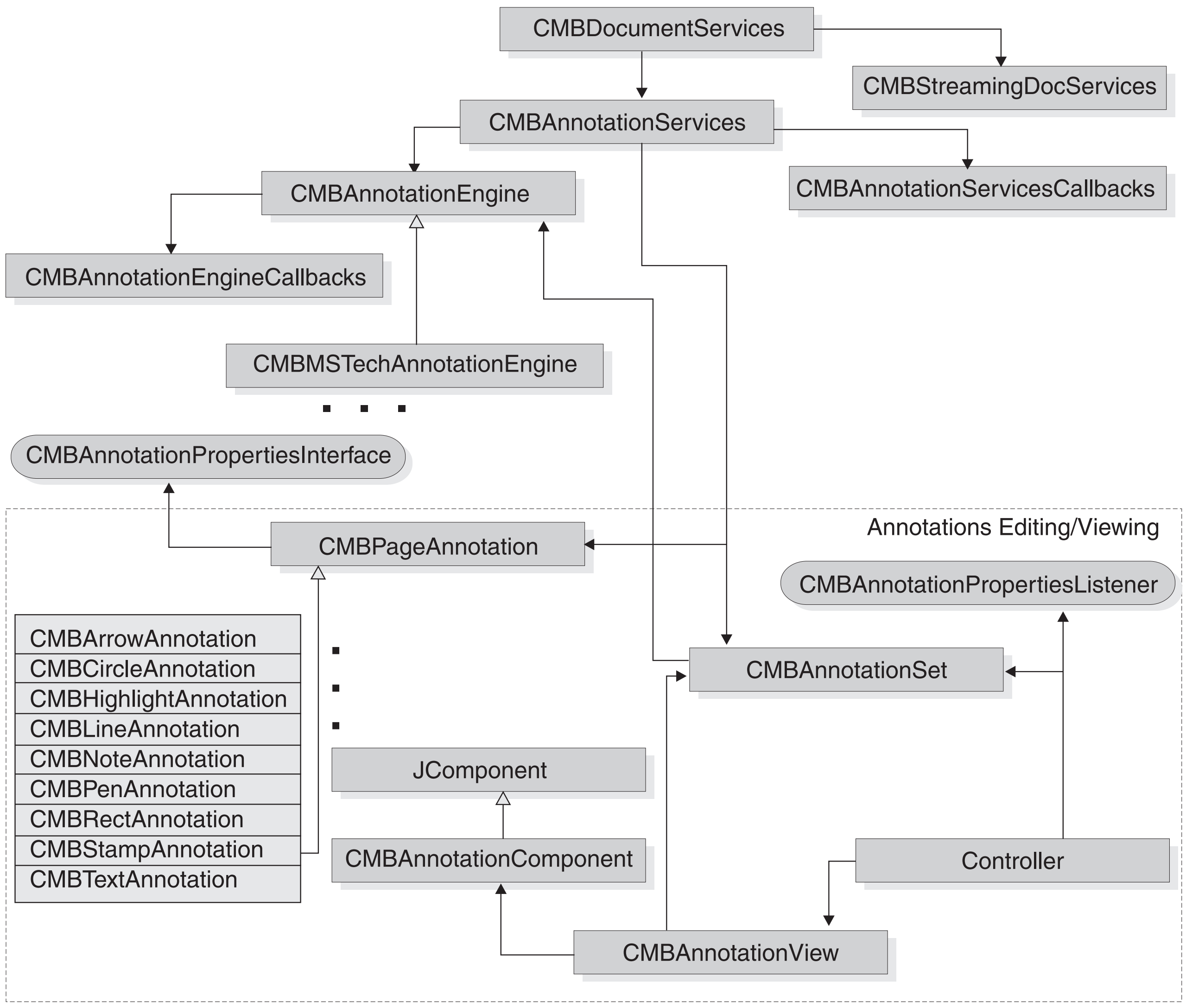


Figure 36. Annotation services class diagram

"Using annotation services interfaces"
"Understanding annotation editing support" on page 612
"Building an application by using the annotation services" on page 613
"Adding a custom annotation type to your application" on page 613

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 by 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.

The following figure 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 the following figure). 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 IBM Information Integrator for Content 8.4 understands only the IBM Content Manager annotation format. This annotation format is used by IBM Content Manager Version 7, IBM Content Manager Version 8.1, IBM Content Manager Version 8.2, IBM Content Manager Version 8.3, IBM Content Manager Version 8.4, and VisualInfo for AS/400 backend servers.

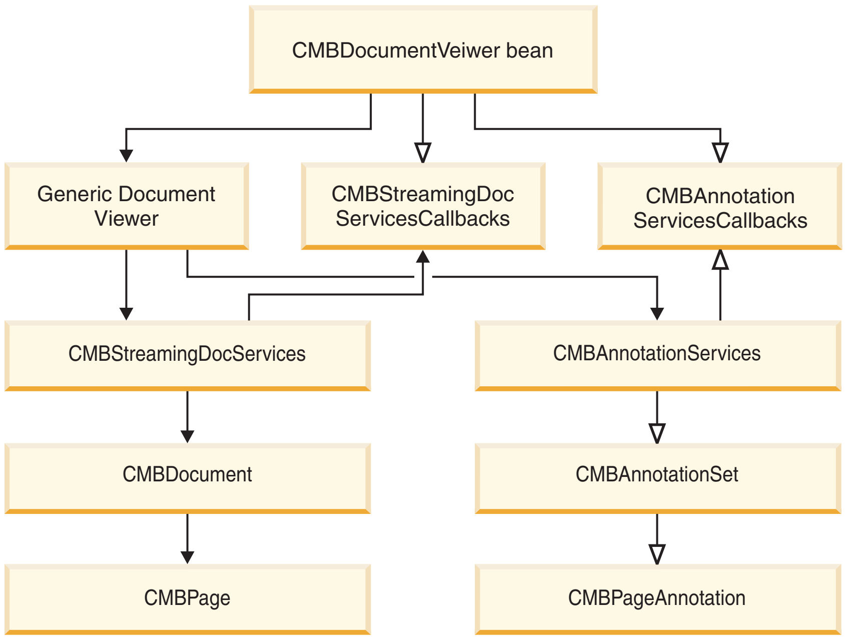


Figure 37. Annotation services and generic document viewer association

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 IBM Content Manager annotation types that you can create: `CMBArrowAnnotation`, `CMBCirlceAnnotation`, `CMBHighlightAnnotation`, `CMBLineAnnotation`, `CMBNoteAnnotation` , `CMBPenAnnotation`, `CMBRectAnnotation`, `CMBStampAnnotation`, and `CMBTextAnnotation`.

Note: You can use annotation services in non-GUI applications.

Building an application by using the annotation services

To build an annotation services application, you have to create a subclass of `CMBAnnotationServicesCallbacks`, create an instance of `CMBAnnotationServices`, and then get an instance of `CMBAnnotationSet`.

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 window. The properties window 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 the Java document viewer toolkit

The following tables summarize the document types and conversions supported by default (by using the supplied conversion engines).

Additional conversion capabilities can be plugged in by creating classes that extend `CMBDocumentEngine` and adding those classes to the list of engines in the engine properties passed to the constructor of `CMBStreamingDocServices`.

For a list of supported document types in the Client for Windows and the eClient, see the related information at the bottom of this page

Document types and conversions on Microsoft Windows

Table 58. Document types on Microsoft Windows

MIME Type	Engine	Paginated	With Annotations	Page Manipulation Supported	Converted To
application/afp	`CMBODDocumentEngine`	Yes[1]	No	No	text/html
application/afp	`CMBODDocumentEngine`	No	No	No	application/afp[2]
application/lin application/ ondemand line	`CMBODDocumentEngine`	No	No	No	text/plain[3]
application/pdf	`CMBSnowboundEngine`	Yes	Yes	No	image/png

Table 58. Document types on Microsoft Windows (continued)

MIME Type	Engine	Paginated	With Annotations	Page Manipulation Supported	Converted To
image/tiff image/tif image/gif image/jpeg image/bmp text/plain application/ vnd.ibm.modcap	`CMBMSTechDocumentEngine`	Yes	Yes	Yes	image/gif image/jpeg image/tiff application/ vnd.ibm.modcap text/plain[5]
image/pcx image/dcx	`CMBMSTechDocumentEngine`	Yes	Yes	No	image/gif image/jpeg
text/enriched text/richtext text/rtf text/html application/ vnd.lotus-1-2-3 application/ vnd.ms-excel application/ wordperfect5.1 application/ msword application/ vnd.lotus- wordpro	`CMBMSTechInsoEngine`	Yes	Yes	No	image/gif image/jpeg
text/url	`CMBJavaDocumentEngine`	No	No	No	text/html[4]

Document types and conversions on AIX and Solaris

Table 59. Document types on AIX and Solaris

MIME Type	Engine	Paginated	With Annotations	Page Manipulation Supported	Converted To
application/afp	`CMBODDocumentEngine`	No	No	No	application/afp[2]
application/pdf	`CMBSnowboundEngine`	Yes	Yes	No	image/png
image/tiff image/tif image/gif image/jpeg image/bmp text/plain application/ vnd.ibm.modcap	`CMBMSTechDocumentEngine`	Yes	Yes	Yes	image/gif image/jpeg image/tiff application/ vnd.ibm.modcap text/plain[5]
image/pcx image/dcx	`CMBMSTechDocumentEngine`	Yes	Yes	No	image/gif image/jpeg
text/url	`CMBJavaDocumentEngine`	No	No	No	text/html[4]

Document types and conversions on Linux

Table 60. Document types on Linux

MIME Type	Engine	Paginated	With Annotations	Page Manipulation Supported	Converted To
application/afp	`CMBODDocumentEngine`	No	No	No	application/afp[2]
application/pdf	`CMBSnowboundEngine`	Yes	Yes	No	image/png
image/tiff image/tif image/gif image/jpeg image/bmp text/plain application/ vnd.ibm.modcap	`CMBMSTechDocumentEngine`	Yes	Yes	Yes	image/gif image/jpeg image/tiff application/ vnd.ibm.modcap text/plain[5]
image/pcx image/dcx	`CMBMSTechDocumentEngine`	Yes	Yes	No	image/gif image/jpeg
text/url	`CMBJavaDocumentEngine`	No	No	No	text/html[4]

Note:

1 The AFP2Web toolkit performs paginated conversion to text/hml. This toolkit is not packaged with IBM Information Integrator for Content; it is available separately. The bridge to AFP2Web contained within `CMBODDocumentEngine` is Windows only; therefore, AFP to HTML conversion is available only on Windows by 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 The text/url MIME type is a special MIME type used to indicate that a plain text document contains a URL. CMBStreamingDocServices converts the URL into an HTML document with an auto-forwarding link.

5 A document of a particular source type can be written only to one of these target types if it is compatible with 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.

Related information

Supported document formats

Working with the page manipulation functions

The Java viewer toolkit provides capabilities for editing image-paged documents at the page level.

Only the CMBMSTechDocumentEngine document engine supports page manipulation. 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 by 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"

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. IBM Content Manager provides the following actions:

Action	Description
new_doc	Creates a document. The document created within the viewer initially contains no pages. Saving the document creates it on the server.
save_as	Saves the document in the viewer as a new document on the server.
export_doc	Exports the document displayed in the viewer to the file system. This action displays a dialog that allows the user to specify the file name, type, and directory location. A check box enables imprinting the annotations onto the document pages.

3. Create a handler for implementing the `CMBGenericDocSaveEvent` listener. This handler is registered with the instance of `CMBGenericDocViewer`. It gains control whenever a calling viewer action requires saving. The handler also gains control when saving the document when the document is closed.
   ```
   class SaveHandler implements CMBGenericDocSaveListener
   {
   public void genericDocSave(CMBGenericDocSaveEvent evt)
   {
      CMBDocument document = evt.getDocument();
       CMBAnnotationSet annoSet = evt.getAnnotationSet();
       if (evt.getSaveAsNew()) {
         .. logic to save new documents
   ```

```
    }
   if (document.isModified()) {
  OutputStream docstream = ..
        docServices.setPreferredFormats(
          new String[] { document.getMimeType()});
        document.write(docstream);
      }
      if (annoSet.isDirty()) {
        OutputStream annostream = ..
        annoSet.write(annostream);
      }
      ..other save logic
      document.setModified(false);
      annoSet.setDirty(false);
     }
  }
```

Typically, performing page manipulation operations on documents with annotations modifies the annotations. Analyze the impact of updating a document and its annotations to avoid updating the document and annotations separately if the operations fail.

A `CMBGenericDocSaveEvent` handler should save both annotations and document content. Prior to V8.3, users used callbacks to save annotations. Callbacks are used when the `isDirty()` flag for a `CMBAnnotationSet` is true and either there are no registered `CMBGenericDocSaveEvent` handlers or the registered handlers do not reset the `isDirty()` flag on the annotation set.

If the `CMBGenericDocSaveEvent` handler saves annotations in all cases, the saving logic in the callbacks is optional.

4. Add logic to enable or disable page manipulation capabilities according to user privileges. The following methods enable or disable page manipulation capabilities in `CMBGenericDocViewer`:

Method	Description
`setPageManipulationEnabled`(boolean enable)	Enables or disables all page manipulation capabilities in the viewer.
`setPrivilege`(CMBDocument document, int privilege, boolean enable)	This method can enable and disable page manipulation on individual documents, and enable and disable the ability to create documents.

5. Add logic to respond to page manipulation operations within the viewer. This is usually necessary if there is additional information that is maintained by the application related to document pages. When users add or remove pages, the application must synchronize the page-specific information with the viewer information.

   Deleting pages in a document does not destroy those pages. Deleting removes pages and places them into a removed pages object. Performing a cut operation places a removed pages object on the clipboard. Deleting places this removed pages object on the tasks performed queue, for use in an undo operation. To allow saving application-specific information, `CMBGenericDocStateChangedEvent` provides `getRelatedInfo` and `setRelatedInfo` methods, which allows an application to associate a serializable object with pages deleted or copied (by using `setRelatedInfo`) and retrieved when the pages are added (by 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

IBM 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 objects, 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 `CMBObject` Java bean. This can be translated to a Java character encoding by 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 window 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 are displayed.

The Preferences window includes the following buttons:

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 window.

Cancel
: Closes the window and restores all preferences that have not been applied to their values at the time the window was opened.

You can change the configuration of pages and preferences on each page by 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 sees only 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 preferences 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 Microsoft 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`.

Preferences support is provided by the `CMBGenericDocViewer` class by 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*.

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** check box on the Preferences window, use the **Preferences** tool shortcut key, ctrl+shift+P to display the Preferences window.

Print preferences

Clicking **Print** in the tree view shows the general print preferences.

You specify the settings on the Print window by using the new printing preferences. These preferences include printing a document with or without annotations, and scaling and centering the document page. Use the Preferences window to change 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 window.

Printing.centerOnPage
Controls the center on page check box on the Print window.

Printing.fitToPage
Controls the fit to page check box on the Print window.

Printing.zoom
Controls the zoom percent on the Print window. This preference 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. For more information, see the *Application Programming Reference*.

Printing

You can print multiple documents in one step. The **Print All Documents** viewer action button prints all documents that are currently open.

You specify the settings on the Print windows by 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 change 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 so many documents are printed, printing multiple documents queues the rendering of documents in order to manage memory usage.

New classes have been added for gathering feedback on printing progress. The classes `CMBGenericDocPrintStatusEvent` and `CMBGenericDocPrintStatusListener` are used to gather feedback on the printing progress. For more information, see the *Application Programming Reference*.

Document indicator preferences

Clicking **Document Indicator** in the tree view provides the following:
- The 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 and color. A timer is useful because a document might be briefly viewed only when navigating from one document to another.
- Control over the font style, color, and time of delay in the viewer preferences.

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 for the following setting: `Appearance.tab.indicate=true`. The default value is bold. The valid values are bold, italic, bold_italic, and plain.

Appearance.tab.color
: Specifies the color of the text for the document tabs for documents that have been viewed. This preference for the following setting: `Appearance.tab.indicate=true`. The default value is green (#00ff00). The valid values are hex strings that represent RGB values as #rrggbb.

Appearance.tab.delay
: Specifies how many seconds a document must be viewed before the tab is indicated. This preference is used for the following setting: `Appearance.tab.indicate=true`. The default value is 10. The valid values is an integer from 0 to 1024.

For more information, see the `CMBGenericDocViewer` methods, `hasDocumentBeenViewed` and `setDocumentViewed`, in the *Application Programming Reference*.

Annotation preferences

Clicking **Annotation** in the tree view shows the general annotation preferences.

Clicking an individual annotation type displays the preferences for that annotation type.

Java viewer layout tools

To enable displaying of 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 view 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 by using the drop-down lists. The side-by-side layouts display two views onto the same set of documents in the viewer. Annotations or page manipulations performed in one pane 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 tab pane and jump to the pages by double-clicking them. Shortcut key: Ctrl – 9.

The Java viewer toolkit enables side-by-side viewing. To enable side-by-side viewing, see the `CMBGenericDocViewer`, `CMBAnnotationServices`, `CMBAnnotationView`, `CMBPageAnnotation`, `CMBAnnotationSet`, and `CMBAnnotationSelectedEvent` methods in the *Application Programming Reference*.

To define toolbars and menus, the following layout tools are included in the viewer configuration properties:

layout_tabbed
Action to switch to tab 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 by 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 by 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 by using the keyboard keys. If you have JAWS open, move the annotations by 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 by 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

IBM 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 IBM 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.

Review the information about installing the tag library and servlet, configuring them, and the information about building WAR/EAR files.

Related reference

Adding the tag library

Using the tag library

A JSP sample uses 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><b>Available Search Templates</b></td>
          </tr>
          <cmb:searchtemplates>
      <tr>
              <td><%= searchtemplate.getName() %></td>
          </tr>
          </cmb:searchtemplates>
    </table>
    </body>
</html>
```

The `taglib` directive declares that the page uses the IBM 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. The following table 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 61. Tag library conventions

Scope	Name	Type	Description
application	connectionPool	`CMBConnectionPool`	The connection pool bean that is shared across sessions
session	connection	`CMBConnection`	The instance of `CMBConnection` for the session
session	schema	`CMBSchemaManagement`	Schema management bean
session	data	`CMBDataManagement`	Data management bean
session	user	`CMBUserManagement`	User management bean
session	query	`CMBQuesryService`	Query service bean
session	traceLog	`CMBTraceLog`	Trace log bean; all of the other beans send their trace to this bean
session	docservices	`CMBDocumentServices`	Document services bean
request	item	`CMBItem`	The last item that you operated on
request	items	`CMBItem[ ]`	Collection of the items that you last operated on
request	searchTemplate	`CMBSearchTemplate`	The selected search template
request	searchResults	`CMBSearchResults`	The results from the last search

Tag summary

The tag library consists of the connection related tags, search related tags, schema related tags, item related tags, document related tags, and folder related tags.

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

The schema related tags iterates through search templates that are available, search criteria of a search template, operators used in a search, values of search criterian, federated attributes and entities.

<cmb:searchtemplates searchTemplates="searchTemplate"> ... </cmb:searchTemplates>
This tag iterates through the available search templates.

searchTemplates
Specify the name of an array of type `CMBSearchTemplate[]` to contain the search templates.

searchTemplate
A variable of type `CMBSearchTemplate` to contain the search template.

<cmb:searchcriteria searchTemplate="searchtemplate">criterion ... </cmb:searchcriteria>
This tag iterates through the search criteria of a search template.

searchTemplate
Specify the name of the search template.

criterion
A variable of type `CMBSTCriterion` to contain the search criterion.

<cmb:displaycriteria searchTemplate="searchTemplate">criterion ... </cmb:displaycriteria>
This tag iterates through the search criteria that can be displayed for a search template.

searchTemplate
Specify the name of the search template.

criterion
A variable of type `CMBSTCriterion` to contain the search criterion.

<cmb:allowedoperators criterion="criterion">operator ... </cmb:allowedoperators>
This tag iterates through the operators allowed for a search criterion.

criterion
Specify the name of a variable of type `CMBSTCriterion` to contain the search criterion.

operator
A string to contain the value of the operator.

<cmb:predefinedvalues criterion="criterion"> value... </cmb:predefinedvalues>
This tag iterates through the predefined values of a search criterion.

criterion
Specify the name of a variable of type `CMBSTCriterion` to contain the search criterion.

value A string to contain the predefined value of the search criterion.

<cmb:entities schema="schema">entity ... </cmb:entities>
This tag iterates through the available federated entities.

schema
Specify the name of a variable of type `CMBSchemaManagement` containing the schema.

entity A string to contain the name of the entity.

<cmb:attributes entity="entity" schema="schema">attribute ... </cmb:attributes>
This tag iterates through the federated attributes of a federated entity.

schema
Specify the name of a variable of type `CMBSchemaManagement` to contain the schema.

entity Specify the name of the entity.

attribute
A string to hold the name of a variable of type `CMBAttribute` that contains the attribute.

Search related tags

The search related tags iterate through search results.

<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

You can use item related tags to iterate through attributes, contents, notelogs, privileges, resources, and view of an item.

<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

The folder related tags helps you iterate through contents of a folder.

<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

The document related tags iterates through currently loaded documents and pages of a document.

<cmb:viewerdocuments docservices="*docservices*">document ... </cmb:viewerdocuments>
This tag iterates through the documents that are currently loaded.

docservices
Specify the name of a variable of type `CMBDocumentServices`.

document
A variable of type `CMBDocument` to contain the document.

<cmb:documentpages document="*document*"> docpage... </cmb:documentpages>
This tag iterates through the pages of a document.

document
Specify the name of a variable of type `CMBDocument` to contain the document.

docPage
A variable of type `CMBPage` to contain the page.

IBM Information Integrator for Content controller servlet

IBM 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 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

Some of the aspects of the servlet that you can use are connection pooling, logon of timed-out session, clean up after session termination, setting up of the locale correctly, and use of different JSP sets.

Some of the aspects of the servlet that you can use are:

Connection pooling
: The controller servlet uses IBM Information Integrator for Content connection pooling to provide high performance connection management. The time in which a connection is allocated to a session might 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 IBM 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 `cmbservletjsp.properties`, by default, describes the JSPs to use for responses to servlet actions. The location of the properties file is an application parameter. Therefore, several different web applications could be written by using different sets of JSP.

Extending the servlet
: All actions known to the servlet are defined in a properties file named `cmbservlet.properties` (default). You can add, modify, or delete servlet actions by changing this file. To add a new action, follow these steps:

1. Implement a class to perform the action. The class must extend `com.ibm.mm.servlets.CMBServletAction`.
2. Add the name of the class and the action name to the `cmbservlet.properties` file. This has the following syntax:

```
actions = list of actions
action.<action_name>.class = class_name
```

`actions` lists the actions understood by the servlet. For each action, a line within the properties file defines the class for the action. For example, to add an action named `replay`, in a class named `ReplayAction`:

```
actions =... replay
action.replay.class = ReplayAction
```

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"

Servlet reference

You use a set of application parameters, request parameters, and a properties file to use the controller servlet in your applications.

"Conventions"

"Application parameters"

"Properties file" on page 636

"Request parameters" on page 636

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 theIBM Information Integrator for Content tag library.

Application parameters

The servlet understands the following application parameters (an alternative is to place these in the `cmbservlet.properties` file).

Application parameter	Values	Description
servletPropertiesURL	URL	The location of the `cmbservlet.properties` file
defaultServerType	Fed, ICM, OD, V4, IP, ...	Default logon information. This, along with **defaultServer**, **defaultUserid**, and **defaultPassword** can be used in situations of shared user ID. Rather than prompting with a login page, the default logon information will be used to perform the logon.
defaultServer		Default logon information.
defaultUserid		Default logon information.
defaultPassword		Default logon information.
connectionpool	Boolean: TRUE | FALSE	To enable connection pooling
maxfreeconnection	integer	Maximum number of connections available in a connection pool.

Application parameter	Values	Description
minfreeconnection	integer	Minimum number of connection available in a connection pool
timeout	integer	The time duration (in milliseconds) after which a free connection will be disconnected and destroyed.
noSessionPage	URL	This is a page to display for logon, if the servlet is invoked without an established session or connection. This can be used to prompt for logon and chain back to the original action, allowing bookmarked links into IBM Information Integrator for Content to work even if the user must log on.
timedOutPage	URL	This is a page to display if the session has timed out due to inactivity.
serverErrorPage	URL	This is a page to display if an error has occurred in accessing a server.
connectFailedPage	URL	This is a page to display if an error has occurred in connecting to a server. A prompt could be displayed to enter the correct user ID/password for the server and retry can be performed.
tracelevel	0, 1, or 2	To indicate the level of tracing, as follows: • 0 - log nothing • 1 - log exceptions (default) • 2 -log exceptions, alert messages, WebSphere Application Server headers and attributes, IBM Information Integrator for Content ini files, JVM system properties, IBM Information Integrator for Content internal trace information
connectiontype	0, 1, or 2	The location of the IBM Information Integrator for Content database and content server runtimes: • 0 - local (default) • 1 - remote • 2 -dynamic
cmbclient	URL	Location of `cmbclient.ini`
cmbcs	URL	Location of `cmbcs.ini`
serviceconnectiontype	0, 1, or 2	Location of services runtimes • 0 - local (default) • 1 - remote • 2 -dynamic
cmbsvclient	URL	Location of `cmbsvclient.ini`
cmbsvcs	URL	Location of `cmbsvcs.ini`
cmbcc2mime	URL	Location of `cmbcc2mime.ini`
cachedir	name of a directory	Directory to cache documents during document conversion

Application parameter	Values	Description
jnitrace	name of a file	File to which to write the JNI trace information for the JNI logic used in document conversion (for IBM diagnostic purposes)
conversion	Boolean: TRUE or FALSE	If TRUE, documents are converted to formats that can be displayed in a browser on middle tier, if possible. If FALSE, the original document, unconverted, is sent to the browser.
maxresults	integer	Maximum hits returned; -1 (default) means all hits.
valuedelimiter	character	Defines the character that will delimit values in search criteria. The default is locale dependent and is comma (**,**) for US English.
conversion.<mimetype>	<none | document | page >	Conversion options for viewing documents of a specific mimetype. This affects the behavior of the viewDocument servlet. Page means attempt to paginate the document. Document means convert the document to a form readable in a browser. None means perform no conversion -- return the document in its native form.
nameseparator	character	Defines the character that will separate child component attribute from the parent component attribute in qualified names. The default is locale dependent, and is a forward slash (/) for US English.

Properties file

The servlet looks for a properties file, `cmbservlet.properties`. This file defines the actions that the servlet can use, including the actions defined here. It also defines the names of the JSP files that are used.

You can also define the servlet properties on the Web application server (servlet engine). The syntax is the same as used in the file.

The content of `cmbservlet.properties` is stored in a `Properties` object by the control servlet. It can be accessed through the application attribute `cmbServletProperties`, as shown in the following example.

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, based on the action.

  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 timeout 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=<> user ID=<> 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 by using the specified search template and criteria values.

`action=searchEntityentity=<>  {attribute.<attrname>.op=<> attribute.<attrname>=<>}[conjunction=<and|or>]`
: Perform a search by 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 by using `and` (default) or `or`, as specified by the conjunction parameter.

`action=searchQuery queryString=<> {queryParameter.<parametername>=<>}`
: Perform a search by using the specified query string. The query syntax depends on the server being searched.

  Any number of additional query parameters might 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 might be provided.

action=retrieveItem itemId=<>
: Retrieve the attributes and content of an item. This action ensures the latest content that is stored on the server is used.

action=updateItem itemId=<> [entity=<>] {attribute.<attrname>=<attrvalue>}
: Update the attributes of an item. If entity is specified, the item is reindexed. Content is also updated if the servlet is invoked through a post.

action=deleteItem itemId=<>
: Delete an item.

action=addContent itemId=<>
: Add a content part to an item. The content data is posted.

action=getContent itemId=<> contentIndex=<>
: Gets the content part and returns it to the browser.

action=updateContent itemId=<> contentIndex=<>
: Update a content part on an item; the content data is posted. If no content exists, a content part is added.

action=deleteContent itemId=<> contentIndex=<>
: Delete a content part for the specified item.

action=addNoteLog itemId=<>
: Modifies the notelog on an item. The text of the notelog is posted.

action=updateNoteLog itemId=<> notelogIndex=<>
: Modifies the note log of an item; the text of the note log is posted. If a note log does not exist, it is added.

action=deleteNoteLog itemId=<> notelogIndex=<>
: Deletes the note log text of an item. The text of the note log is posted.

Folder related

action=addItemToFolder itemId=<> folderId=<>
: Adds the specified item to the specified folder.

action=removeItemFromFolder itemId=<> folderId=<>
: Removes the specified item from the specified folder.

Document related

action=viewDocument itemId=<>
: Retrieves the document and views it. If the document is paginated, this action forwards to a JSP which generates the viewer frameset. If the document is not paginated, this action returns the actual content of the document.

action=viewPage itemId=<> page=<> scale=<> rotation=<> annotations=<yes|no>
: Retrieves a page of the document.

Servlet toolkit function matrix

The servlet toolkit function matrix helps you determine the actions that are supported on different platforms.

Table 62. 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
`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 63. Servlet function matrix Continued

Action	Fed
`logon`	Y
`logoff`	Y
`searchTemplate`	Y
`searchEntity`	Y
`searchQuery`	Y
`lock`	Y

Table 63. Servlet function matrix Continued (continued)

Action	Fed
`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
`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 IBM 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 IBM 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 by using the log control utility to configure API logging, use the system administration online help.

"Java connector logging"

"C++ connector logging"

"Modifying the logging configuration file for the IBM Information Integrator for Content connectors" on page 643

Related information

Log file locations

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.

Related information

Log file locations

C++ connector logging

C++ has one log manager. C++ references the same log configuration file as Java. However, the IBM 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.

Related information

Log file locations

Modifying the logging configuration file for the IBM Information Integrator for Content connectors

The configuration file is installed in the `cmgmt` directory of the IBM Content Manager working directory. The file is named `cmblogconfig.properties`.

The `cmblogconfig.properties` file contains the following default settings. Do not change these default settings in case the configuration file cannot be found or errors with user-defined settings occur:

- Log4j is set as the default log manager. The default log file name is *user ID*.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 application enabled for IBM Information Integrator for Content 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 log4j.

Option	Description
Section 2 - Default Log Manager Setup	Section 2 has three subsections: • Section 2.1 - Specify Log Priority. Eight priority settings are available. The default priority setting is Error. • Section 2.2 - Log Output Destination Setting. Three settings are available: 1. Log to a file 2. Log to Standard Error 3. Log to Standard Console The default setting is Log to file. • Section 2.3 - Log File Name Setting. Use only when the option in section 2.2 is set to Log to file. The default log file name is *`user ID`*`.dklog.log`. The default location of the log file is *`IBMCMWorkingDirectory`*`/log/connectors.`If any backslashes are used, the log file path must include double backslash (\\) or single slash (/) characters in place of the single backslashes. If you do not specify a path, the log is always located in the current working directory where the application is executed. The log file location is controlled by the following settings: – DKLogOutputFileName for default logger (both C++ and Java) – Log4j.appender.apiAppender.File for log4j 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. The maximum file size by setting DKLogOutputFileSize for default logger, both C++ and Java or log4j.appender.apiAppender.MaxFileSize for log4j.

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.

Troubleshooting text search

IBM Content Manager supports two types of text search: text search of attributes that contain text in components and text search of objects. To debug and monitor common problems with text search, examine the output logs, and take corrective action depending on the error that you are getting.

“Determining the index name and related information of an item type”

“Logging and tracing for text search” on page 646

“Debugging items that are not indexed successfully” on page 649

“Text index is pending and not valid” on page 651

“Rebuilding an index for DB2” on page 652

“Rebuilding an index for Oracle” on page 652

“Text index update stops” on page 653

“Text search does not return results for documents created in certain languages” on page 653

“Receiving an error when updating, reorganizing, or using text indexes for text searchable components” on page 653

“Receiving a search error in the Thai language when database setup is incorrect” on page 654

“Receiving an error when using ICM text search for Thai phrases in DB2” on page 654

Determining the index name and related information of an item type

To debug and monitor text search, you have to first determine the index name and related information of the item type that might cause any text search errors.

To determine the index name and related information:

DB2 Use the following SQL statement:

```
SELECT e.COLNAME, e.INDNAME, C.COMPONENTTYPEID,
e. EVENTVIEWNAME
FROM ICMSTNLSKEYWORDS k, ICMSTCOMPDEFS d,
ICMSTTEXTINDEXCONF c, DB2EXT.TEXTINDEXES e
WHERE k.KEYWORDCLASS = 2 AND
k.KEYWORDCODE = d.ITEMTYPEID AND
d.componenttypeid = c.componenttypeid AND
c.INDEXNAME = e.INDNAME AND
k.KEYWORDNAME='<ITEM TYPE NAME>'
```

The following table shows the sample output for DB2.

Table 64. Sample output for DB2

COLUMNNAME	INDEXNAME	COMPONENT TYPEID	EVENTVIEWNAME
ATTR0000001000	ICMUU01116001001	1116	EVENTIX162822
ATTR0000001003	ICMUU01116001002	1116	EVENTIX182822
ATTR0000001081	ICMUU01117001005	1117	EVENTIX222822
ATTR0000001082	ICMUU01117001006	1117	EVENTIX232822
TIEREF	ICMUU01118001TIE	1118	EVENTIX272822

Oracle Use the following SQL statement:

```
SELECT c.COLUMNNAME, c.componenttypeid, c.INDEXNAME
FROM ICMSTNLSKEYWORDS k, ICMSTCOMPDEFS d, ICMSTTEXTINDEXES c
WHERE k.KEYWORDCLASS = 2 AND
k.KEYWORDCODE = d.ITEMTYPEID AND
d. COMPONENTTYPEID = c.COMPONENTTYPEID AND
k.KEYWORDNAME ='<ITEM TYPE NAME>'
```

The following table shows the sample output for Oracle.

Table 65. Sample output for Oracle

COLUMNNAME	COMPONENTTYPEID	INDEXNAME
ATTR0000001000	1013	ICMUU01013001001
ATTR0000001003	1300	ICMUU01013001002
ATTR0000001008	1014	ICMUU01014001005
ATTR0000001007	1014	ICMUU01014001006
TIEREF	1015	ICMUU01015001TIE

The column INDEXNAME of the row TIEREF contains the text search index name for the corresponding resource. Other rows contain the names of indexes for attribute if the item type has text searchable attributes.

Logging and tracing for text search

DB2 Net Search Extender errors in administrative operations

The administrative operations, such as creating a text searchable item type or altering the characteristics of a text search update index might log three NSE errors at the end of the library server log file, which is ICMSERVER.LOG file by default.

Read the log file to identify the errors or use the same SQL statement to debug more errors if required.

The following example shows a log sample:

```
ICMPLSTI printTEventTableMsg 04173 07/14 07:51:48.156 GMT
;14075146483556cc:11b208dc211:X7fcd ICMADMIN Get Time and Messages from
eventtablename Query Stmt<SELECT TIME, MESSAGE FROM DB2EXT.TEVENTIX414507
ORDER BY TIME DESC FETCH FIRST 3 ROWS ONLY WITH UR> ICMPLSTI
printTEventTableMsg 04224 07/1407:51:48.156 GMT ;14075146483556
cc:11b208dc211:X7fcd ICMADMIN Messases fromeventtableTIMESTAMP:
<2008-07-14-15.51.47.906000> MESSAGE:<CTE0004 Indexupdate ended> ICMPLSTI
printTEventTableMsg 04224 07/14 07:51:48.156 GMT;14075146483556
```

```
cc:11b208dc211:X7fcdICMADMIN Messases from eventtableTIMESTAMP:
<2008-07-14-15.51.47.890000> MESSAGE:<CTE0100 A DB2 operation failed. DB2
information: "42724" "[IBM][CLI Driver][DB2/NT]SQL0444NRoutine
"ICMADMIN.ICMFETCHFILTER" (specific name "SQL080707231230002")
isimplemented with code in library or path "\ICMNLSUF", function
"ICMfetch_Filter" whichcannot be accessed.Reason code: "4".SQLSTATE=42724>
```

DB2 for z/OS OmniFind Text Search Server error in administrative operations

The administrative operations, such as creating a text searchable item type or dropping a text search index might log error at the library server log file, which is `SYSPRINT` by default.

Read the log file to identify the errors. The following example shows a server log sample:

```
ICMPLSCP defineTIEIndex 01961 01/08 20:43:09.630 GMT ;08204038703904
6b:11eb7f97418:X7fe1 ICMADMIN DROP INDEX Index Schema <ICMADMIN> Index Name
<ICMUU01097001001> SQLCODE = -20427, ERROR OCCURRED DURING TEXT SEARCH
ADMINISTRATION STOREDPROCEDURE OF00306E The text search index
'ICMADMIN'.'ICMUU01097001001' does not exist. SQLSTATE RETURN CODE = 38H14
SQL PROCEDURE DETECTING ERROR = DSNXRRTN SQL DIAGNOSTIC INFO = -84700-100
SQL DIAGNOSTIC INFO = X'FFFFFCB1'X'0'X'0'X'FFFFFFFF'X'0'X'0'
```

Tracing in update text index operation

DB2 If you need to debug why objects did not index successfully in text search, you can activate tracing and view the corresponding ICMSTSYSCONTROL.UDFTRACEFILENAME file to see which items failed to index. The records for text index operations resemble the same format as the library server trace records.

Use the **`UDFTRACELEVEL`** environment variable to enable tracing. Trace files are created in **`UDFTRACEFILENAME`** whenever an error occurs regardless of the **`UDFTRACELEVEL`** setting:

- For Windows, the default value of UDFTRACEFILENAME is `%IBMCMROOT%\log\ls\`*`LSDBName`*`\UDFTRACE`.
- For UNIX, the default value is *`IBMWorkingDirectory`*`/log/ls/`*`LSDBName`*`/UDFTRACE`.
- For z/OS, the default value is **`SYSPRINT`**.

Recommendation: Use **`UDFTRACELEVEL`**=15 in most cases.

To enable tracing, set one of the following environment variables:

UDFTRACELEVEL=0
Turns tracing off.

UDFTRACELEVEL=1
Basic Trace. Trace function entry and exit.

UDFTRACELEVEL=2
Traces details except for objects and their conversion output.

UDFTRACELEVEL=4
For ICMFETCHFILTER UDF only, and does not apply to z/OS. Traces the objects retrieved from the resource manager, and creates the following files if applicable:

UDFTRACEFILENAME.**from_preretrieve_exit**
: Contains the current object (if it exists), as returned from the text search user exit.

UDFTRACEFILENAME.**from_rm**
: Contains the current object returned from resource manager.

UDFTRACEFILENAME.**from_postretrieve_exit**
: Contains the current object from the text search user exit after calling resource manager.

UDFTRACEFILENAME.**from_inso_conversion**
: Contains the current object after performing text extraction.

UDFTRACEFILENAME.**after_ucs2_to_utf8_conversion**
: When the database is in Unicode, contains the object after conversion from ucs2 to utf8.

UDFTRACELEVEL=8
: Traces performance for UDF.

UDFTRACELEVEL=16
: Traces the following environment values:
- User name
- Current® directory (not applicable to z/OS)
- Shared LIBPATH (not applicable to z/OS)
- IBMCMROOT (not applicable to z/OS)
- ICMROOT (not applicable to z/OS)
- ICMDEBUG (not applicable to z/OS)

UDFTRACELEVEL=32
: Traces malloc and free.

UDFTRACELEVEL=64
: Traces memory leak in the UDF for dynamic allocated memory usage.

UDFTRACELEVEL=128
: For ICMFETCHFILTER UDF only, and does not apply to z/OS. Traces the indexing status. The library server creates the `*UDFTRACEFILENAME*.itemid` file and stores the last object ID processed before the command fails to identify any object that causes the process to stop when UDF timeout parameter is not set.

`ICMDEBUG` has been deprecated. The trace files for **`ICMDEBUG`** are created under `/tmp` directory for UNIX and `C:\` directory for Windows. For ICMFetchFilter UDF, trace file `icmserver.fetchfilter` is used. For ICMFetchContent UDF, trace file `icmplsud.log` is created.

The **`ICMDEBUG`** values map to the **`UDFTRACELEVEL`** values as follows:

Table 66. Mapping between ICMDEBUG and UDFTRACELEVEL

ICMDEBUG	UDFTRACELEVEL
1	2
2	128
4 (recommended)	15 (recommended)

ICMDEBUG=1
The library server creates the `icmserver.fetchfilter` file (or the `icmplsud.log` file if the ICMFetchContent UDF is used)

ICMDEBUG=2
The library server creates the `itemid.log` file and stores the last object ID processed before the command fails to identify any object that causes the process to stop when UDF timeout parameter is not set.

ICMDEBUG=4 (for ICMFETCHFILTER UDF only)
The library server creates:

icmserver.from_preretrieve_exit
Contains the current object (if it exists), as returned from the text search user exit.

icmserver.from_rm
Contains the current object returned from resource manager.

icmserver.from_postretrieve_exit
Contains the current object from the text search user exit after calling resource manager.

icmserver.from_inso_conversion
Contains the current object after performing text extraction.

icmserver.after_ucs2_to_utf8_conversion
When the database is in Unicode, contains the object after conversion from ucs2 to utf8.

Oracle The text search UDF traces to the library server log file (`ICMSERVER.LOG`). You can also enable the Oracle trace for text search by running the following commands in SQL*Plus:

```
ctx_adm.set_parameter('log_directory', '/tmp/ctxlog');
ctx_output.start_log('logfilename');
ctx_ddl.sync_index('ICMADMIN.ICMUU01010001TIE').
```

You can also read the log file when it is running. It logs every 1000 documents processed.

Related reference

Determining the index name and related information of an item type

Related information

Creating DB2 Universal database user IDs with user rights and privileges

Debugging items that are not indexed successfully

If a document item is not found in the result set of a text search query, the item was not indexed successfully.

To debug the problem:

1. Identify the object ID of the resource that could not be indexed.

   DB2 After Version 8.4.2, UDF captures all of the indexing failure errors into the UDF trace file. The trace file is located according to the UDFTRACEFILENAME column in the ICMSTSYSCONTROL table.

The following sample shows the error that occurs if WebSphere services was not turned on:

```
ICMPLSU3 ICMfetch_FilterNTO              01612 06/19
22:51:55.390 GMT 2009/06/19 15:51:55.390 LCT DB2EXT
;00000000:00007280   DB2EXT IsoSockClientConnect failed
lRC       : 10061    Message   : <Please verify HTTP server and
resource manager are started.>    ItemID    :
<A1001001A09F16B22559A84734>    VersionID : <1>    szTIERef  :
<A1001001A09F16B22559A84734#1#ICMADMIN#
cmi78.svl.ibm.com#CBR.CLLCT001# #/i78rm1/ICMResourceManager#
                  #81920.000000#9080#1#i78ls# #301#1########>
sqlstate : <38C12> ICMPLSU3 ICMFETCHFILTER
00356 06/19 22:51:55.406 GMT 2009/06/19 15:51:55.406 LCT DB2EXT
;00000000:00007280   DB2EXT Exit rc=7064 reason=0 extrc=0
extreason=0
```

Alternatively, you can query the Net Search Extender event table. To list the errors, use the following SQL statement:

```
SELECT * from DB2EXT.EVENTIXxxxxx
```

where `EVENTIXxxxxx` is the index name.

The following table shows the sample output for DB2.

Table 67. Sample output for DB2

OPERATION TIME	SEVERITY	REASON	MESSAGE
2008-09-12-15.05.59.266001	1	3	`CTE0003 Index update started`
2008-09-12-15.06.01.095000	1	5	`CTE0005 Index update commit: "1","0","0" documents inserted, updated, and/or deleted successfully.`
2008-09-12-15.06.01.095001	1	4	`CTE0004 Index update ended`
2008-09-12-15.06.45.126002	1	3	`CTE0003 Index update started`
0 2008-09-12-15.06.51.204000	8	100	`CTE0100 A DB2 operation failed. DB2 information: "38C12" "[IBM][CLI Driver][DB2/NT] SQL0443N Routine "ICMFETCHFILTER" (specific name "SQL080602175028101") has returned an error SQLSTATE with diagnostic text "IsoSockClientConnect rc = 10061". SQLSTATE=38C12`
2008-09-12-15.06.51.204001	1	4	`CTE0004 Index update ended`

If the messages do not contain the item ID, enable the traces by using ICMDEBUG to retrieve the error and the problem item ID.

Oracle Use normal `ICMSERVER.LOG` trace (-15) to retrieve the item IDs that caused the error.

The following example shows the sample output from the trace:

```
ICMPLSFC generateToken 03595 09/16 00:18:28.000
GMT Input to generateToken
TokenDuration(RM): 172800
```

```
ICMPLSFC generateToken 03641 09/16 00:18:28.000
GMT Input to generateToken
Attr – version: 2
Attr – token case: 0
Attr –mode: 1
Attr – type: 0
Current time: 1221524308
TokenDuration(actual): 172800
ObjectID:<A1001001A08I15B71046I728101>
Exp time: 1221697108
```

2. DB2 Debug the text extraction step on the object ID of the resource that could not be indexed. Check whether the text extraction step was successful by using the following SQL statements:

   For ICMFetchFilter UDF:
   ```
   SELECT ICMADMIN.ICMFETCHFILTER(TIEREF)
   from ICMUTxxxxx001 where RTARGETITEMID='ObjectID';
   ```

   For ICMFetchContent UDF:
   ```
   SELECT ICMADMIN.ICMFETCONTENT(TIEREF)
   from ICMUTxxxxx001 where RTARGETITEMID='ObjectID';
   ```

   This SQL statement does not print the output because UDF returns a LOB. However, if it returns successfully, the text extraction process was done successfully. If it does not return, you can trace it or look at the error messages in the `SELECT * from NSE table DB2EXT.TEXTINDEXES` to determine the problem.

3. Re-index the object. To re-index the object in the next text update index operation, type the following SQL statement:
   ```
   UPDATE ICMUTxxxxx001 SET TIEREF=TIEREF WHERE
    RTARGETITEMID='<Object ID>';
   ```

   where `xxxxx` is the five digits that contain the component type ID and `<ObjectID>` is the string (26 characters) part ID from step 2.

Related reference

Determining the index name and related information of an item type

Text index is pending and not valid

Pending text index means that there are items that are waiting to be indexed. If you check for pending index, it might help determine whether an update index operation has stopped.

To check whether text index is pending:

For DB2

Query the event table to check whether the indexing process has finished by using the following SQL statement.

```
SELECT TIME,MESSAGE from DB2EXT. EVENTIX272822
```

The following table shows the sample output for DB2.

Table 68. Sample output for DB2

Time	Message
2008-06-11-13.56.13.195000	`CTE0003 Index update started`
2008-06-11-13.56.24.867000	`CTE0005 Index update commit: "1","0","0" documents inserted, updated, and/or deleted successfully.`

Table 68. Sample output for DB2 (continued)

Time	Message
2008-06-11-13.56.24.867001	`CTE0004 Index update ended`

If you get the last message, `CTE0004 Index update ended`, the indexing process is complete. If the indexing process has stopped, you will get the first message only, `CTE0003 Index update started`, but not get the message `CTE0004 Index update ended`. If the indexing process has stopped, see the instructions to resolve when updating text index stops.

For Oracle

Query the event table by using the following SQL statement:

```
select owner,index_name, status from dba_indexes
where index_name like 'ICM%';
```

The following table shows the sample output for Oracle.

Table 69. Sample output for Oracle

Owner	INDEX_NAME	Status
ICMADMIN	ICMUU01001TIE	VALID
ICMADMIN	ICMUU01015TIE	INVALID
ICMADMIN	ICMUU01009TIE	INPROGRS

In the previous sample output table:

- If the status of the index is INVALID, the index must be rebuilt.
- If the status of the index is INPROGRS (in progress), an update index operation is running. If the corresponding pending count is not decreasing, the update text index operation might have stopped.

Related reference

Text index update stops

Rebuilding an index for DB2

To rebuild an index:

1. Manually drop the index that you want to rebuild by using ManuallyDropDb2CMTextindexes.SQL script.
2. Enable text search for the item type again by using the system administration client.

   Important: The next time update index is run, it causes all selected item in the script to get reindexed.

Rebuilding an index for Oracle

To rebuild an index:

Alter index ICMADMIN.ICMUU01010001TIE REBUILD

Text index update stops

When text index update has not completed, the best action is to recycle the database manager. Identify the item ID that causes the updating process to stop. Try to create the problem again by following the steps in the related information. If the problem can be created again, retrieve the objects into a file and contact IBM Software Support for further help. For DB2 users, enable the UDF timeout by setting the UDF timeout value by using the system administration client so that items similar to this are skipped and logged.

Related reference

Debugging items that are not indexed successfully

Text index is pending and not valid

Text search does not return results for documents created in certain languages

If text search does not return results for documents in certain languages, you must enable text search in all languages by creating a LEXER preference that is appropriate for the language of the documents.

Symptoms

Text search does not return results for documents created in certain languages.

Resolving the problem

You must enable text search on non-European or white-space delimited languages, by specifying the LEXER preference or by using CTX_SYS.BASIC_LEXER.

Create a LEXER preference that is appropriate for the language of the documents in the index. On Oracle, you must do this manually, and then select the LEXER preference when you are creating the index.

To index Japanese documents, you must create a LEXER preference by completing the following steps:

1. Connect to the database as an IBM Content Manager administrative user. Example that uses sqlplus::`sqlplus user ID/password@dbname`
2. Create the preference: `exec ctx_ddl.create_preference('MY_JAPANESE_LEXER','japanese_lexer');`
3. The preference you created is displayed in the list in the system administration client. You must close the system administration client and then restart the system administration client again. Under LEXER, select the preference you created when you create the index.

Receiving an error when updating, reorganizing, or using text indexes for text searchable components

If you receive an error when you are updating or reorganizing the text indexes, you must reset the DB2 Net Search Extender password.

Symptoms

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]
```

Causes

The DB2 Net Search Extender (DB2 UDB) password is either not set or set incorrectly.

Resolving the problem

Reset the password in the IBM 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.

Receiving a search error in the Thai language when database setup is incorrect

If the database and the text index is not created in a specific way, the search for Thai phrases in IBM Content Manager results in an error.

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.

Resolving the problem

To support phrased searches in Thai content in IBM 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:
`/MyItemTypeView[contains-text(ICMPARTS/@TIEREF, " 'Thai word' ")=1]`

To search for Thai phrases by using the DB2 Thai-specific advanced text search syntax: `/MyItemTypeView[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

When you use ICM text search for Thai phrases, you get an error because the basic text search supports one Thai word at a time. To search Thai phrases, use advanced text search.

Symptoms

When you use the IBM® WebSphere Application Server Windows client, or the WebSphere Application Server APIs to text search for Thai language phrases, you do not always get the expected results back when the database backend is DB2.

Causes

The default query string that is generated by the WebSphere Application Server Windows client for text search is in the following format:
`/MyItemTypeView[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.

Resolving the problem

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). `"/MyItemTypeView[contains-text-db2(ICMPARTS/@TIEREF, \" IS ABOUT TH_TH 'Thai phrase' \")=1]"`

If you are using the WebSphere Application Server 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*.

Related reference

DB2 Net Search Extender Help Overview

Receiving an error when compiling C++ applications that are Unicode enabled

If you get an error when you are compiling C++ applications that are Unicode enabled, disable the Unicode flag to resolve the problem.

Symptoms

You are getting an error when compiling C++ applications that are Unicode enabled.

Causes

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

To store objects in an IBM Content Manager Unicode enabled database by 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 IBM Content Manager C++ APIs can handle ASCII byte strings by 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 IBM Content Manager 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).

Resolving the problem

When compiling C++ applications that use the IBM Content Manager Version 8 C++ APIs, you must compile with the Unicode flag set to OFF.

The IBM Content Manager product installs by 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

When you are adding or updating an item, you get an error because you are using reference attributes to connect with an item of same item types.

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'`.

Causes

In general, you cannot use reference attributes to connect with an item of the same item type. References are meant to reference items of different item types. You can create a reference to an item of the same item type by setting and modifying the reference attribute on an item that is persistent. You must make persistent the modifications to the reference attribute value only.

An unexpected SQL error with SQL return code of -910 occurs if changes relating to the reference attribute are not made independently of other changes.

Resolving the problem

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

You get an error when you run an application by using Java beans when WebSphere Application Server connection pooling is enabled.

Symptoms

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: DSRA9350E: Operation Connection.commit
is not allowed during a global transaction.
```

Causes

The IBM Content Manager API does not support Java beans when WebSphere Application Server connection pooling is enabled.

Resolving the problem

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 IBM Content Manager APIs.

Cannot add, store, retrieve, or update a resource item

You cannot add, store, or update an item in the resource manager because the resource manager is not running or available. You must verify that the resource manager is running before adding or updating a resource item.

Symptoms

You get errors when you attempt to add, store, retrieve, or update resource items (items that are stored in the resource manager) or IBM 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.

Resolving the problem

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/icmrm/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

If you try to import a DKDDO object from XML and you get an exception message, you must check that the date, time, or timestamp values in the XML files have the correct formats.

Symptoms

You are receiving an `IllegalArgumentException` with the message `DGL0303A: Invalid parameter` when you import a `DKDDO` object from XML.

Causes

You are using the wrong date, time, or timestamp format.

Resolving the problem

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)
```

Cannot download ICMBASESTREAM part type content

You cannot download the ICMBASESTREAM part type content unless you have defined the IBM Content Manager VideoCharger collection on the server.

Symptoms

If you create a document item type and add an ICMBASESTREAM part type to that document, you must have already defined an IBM Content Manager VideoCharger collection on the server or you cannot download the content. If you attempt to download the content, you receive the following error: `R com.ibm.mm.beans.CMBException: DGL3980A: The resource manager returned an invalid response`

Resolving the problem

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.

Document routing: Cannot select action icon when using Oracle

You cannot create an action icon in Oracle because the action icons are not supported in Oracle database.

Symptoms

You cannot create a custom document routing action in Oracle.

Resolving the problem

In the system administration client, you can create an action that represents custom tasks to be completed in client applications. When you create an action, you can select a custom icon to represent the icon in the user client.

If your IBM Content Manager Version 8.3 library server database is on Oracle, the user action icon is not supported. The action icon is binary data that is associated with a document routing `DKWorkFlowActionICM` object. The intended usage is to display actions by using custom icons in your custom client application. However, if your library server database is on Oracle, you cannot store or retrieve the action icon in IBM Content Manager Version 8.3.

Decision point fails after importing from an XML file

The decision point in document routing fails after import operation because the item type identifier generated by the target library server is different from the original one.

Symptoms

In document routing, the decision point fails after importing the workflow information from an XML file.

Causes

If item types are referenced in a decision point, the decision point no longer works after importing the workflow information from an XML file that was created in a different library server.

Even though the item type has the same name after the import process is finished, the item type identifier generated by the target library server is probably different from the one in the original library server. However, the decision point still references the original item type identifier, and the tables for these item types does not exist.

Resolving the problem

Modify the decision expression in an existing decision point branch that references to an item type. After you save the change, the decision point branch is created again.

Update index operation fails with DGL5200A if UDF timeout is enabled

If the UDF timeout is enabled, the update index operation fails in Linux or Linux for System z operating systems.

Symptoms

On Linux or Linux for System z operating systems, if the UDF timeout is enabled with a positive value, an error occurs when you update the index of the text search enabled item types. The stack trace looks like:

```
com.ibm.mm.sdk.common.DKUsageError:
DGL5200A: Error defining or changing text index on component type 1042.
Text search return code = 192.;
ICM7064: An unexpected error occurred when the library server was requesting
that the text search service create, update, or delete a text index.
See the library server log for more details.
For DB2, use the external return code from library server to
check the text search error message in the text search documentation.
Correct the problem and try again.
For other databases, see the relevant text search documentation.
(STATE) : [LS RC = 7064, SQL RC = 192]
        at com.ibm.mm.sdk.server.
         PItemTypeDefImpICM.defineTextIndex(PItemTypeDefImpICM.java:7951)
        at com.ibm.mm.sdk.server.
         PItemTypeDefImpICM.updateTextIndexes(PItemTypeDefImpICM.java:7781)
        at com.ibm.mm.sdk.server.
         PDatastoreDefCacheICM.updateTextIndexes(PDatastoreDefCacheICM.java:4164)
        at com.ibm.mm.sdk.server.
         PDatastoreDefImpICM.updateTextIndexes(PDatastoreDefImpICM.java:928)
        at com.ibm.mm.sdk.common.
         DKDatastoreDefICM.updateTextIndexes(DKDatastoreDefICM.java:1461)
```

Resolving the problem

Modify your *DB2LIBPATH* variable setting in the `profile.env` file. Put the 32-bit `lib` directory of your instance in the *DB2LIBPATH* variable. For example, if your instance is at `/home/db2inst1`, add `/home/db2inst1/sqllib/lib` to your *DB2LIBPATH* variable and restart the instance.

Tip: If your have a 64-bit instance, add `/home/db2inst1/sqllib/lib32` instead of the previous path because in the 64-bit instance, the `/home/db2inst1/sqllib/lib` path is linked to the 64-bit lib directory by default, so specify the actual 32-bit lib directory as `/home/db2inst1/sqllib/lib32`.

DKString DKLobICM::getContent () const not recommended for ICM connector

You should use the correct `getContent` method with the `DKByteArray*` return type for ICM connectors.

Symptoms

You get incorrect results when you are using `DKString DKLobICM::getContent() const` method with ICM connector.

Causes

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 such as `substring()` on the instance, you might receive incorrect results. You should not use binary data with a `DKString` class.

Resolving the problem

Use the `DKByteArray* DKLobICM::getContent()` method.

Cannot load user exit program

The user exit program cannot be loaded because the *$ICMDLL* variable is not set to the correct location.

Symptoms

You cannot load your user exit program.

Causes

You might not have set the *$ICMDLL* variable correctly.

Resolving the problem

The user exit program must be located in the `$ICMDLL/ICMNLSDB/DLL` directory, where *ICMNLSDB* is the name of the library server.

You must also complete the pre-installation steps to set the *$ICMDLL* environment variable on the IBM Content Manager instance. You can find the steps in the IBM Information Integrator for Content information.

To point *$ICMDLL* variable to the correct location:
1. Connect to the library server database.
2. Select `pathicmdll` from `icmstsyscontrol`.
3. Update `db2InstHome/sqllib/userprofile`:
   ```
   ICMDLL= <value from #2 above>
   export ICMDLL
   ```
4. Run `recycle db2 instance`.
5. Place the following library server user exit routines 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

Some administration APIs cannot be called in an explicit transaction because of the underlying database restrictions.

Symptoms

If you call the specific APIs, you receive the following exception: `DKUsageError DGL0594A: This operation cannot be performed within an explicit transaction.`

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 IBM Content Manager systems. The item type restriction also applies to Oracle and the restrictions for the user methods apply only to IBM Content Manager systems that use Oracle.

Connect method does not check for null or empty password in Trusted Logon mode

When the Trusted Logon mode is enabled in IBM Content Manager, the applications can log in to the library server by using the `connect` method without providing a password.

Symptoms

IBM Content Manager client applications that use the `connect` method can authenticate user IDs without a password whenever Trusted Logon mode is enabled.

Causes

When the IBM Content Manager server is set up for Trusted Logon mode, any client application that uses the normal `connect` method can log in to the library server without providing a password because it assumes that the WebSphere Application Server performs the authentication. The `connect` method does not check for a null or empty password. After the method passes the database authentication layer (by using the shared database connection ID in the `INI` file), the method calls the logon without a password. Therefore, when Trusted Logon mode is enabled, IBM Content Manager allows the logon to succeed.

Resolving the problem

Always use the `connectWithCredential` method with Trusted Logon to force authentication through the WebSphere Application Server. The `connectWithCredential` method 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

If you search the z/OS library server database by using wildcard characters, you can get incorrect results.

Symptoms

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:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
X     !!! Exception !!!    X
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
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)
```

Resolving the problem

To fix this problem, apply the DB2 Universal Database z/OS PTF that corresponds to the level of DB2 Universal Database that you have by using:
- PTF UK08338 (DB2 UDB zOS Release 710)
- PTF UK08339 (DB2 UDB zOS Release 810)

Receiving a DKException message when query times out

You will get an exception message when the query times out because your IBM Content Manager application sends a query timeout value with your IBM Content Manager queries.

Symptoms

When the query times out, you receive a DKException message.

Causes

Your IBM Content Manager application is sending a query timeout value as one of the query options with all the IBM Content Manager queries.

Resolving the problem

When an IBM Content Manager query times out, the `DKDatastoreICM` query method throws a `DKQueryException` message with an error ID of `DKMessageICM.DK_ICM_MSG_QUERY_TIMEOUT (7254)`.

See the following Java example for 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 specified 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

To display an Excel document properly in the Java viewer, you must disable the document protection feature in Excel.

Symptoms

If you have document protection enabled in an Excel document, the document will not display properly.

Causes

The Java viewer does not support data protection and encryption feature available in some Microsoft Office applications.

Resolving the problem

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

All UTF plain text documents are left-aligned in Java viewer. To display the documents that are in Arabic or Hebrew that are right-aligned, use Arabic or Hebrew encoding instead of UTF-8 encoding.

Symptoms

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.

Resolving the problem

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

The line wrapping in the Java viewer causes graphical annotations to be placed incorrectly relative to the text. You can fix the misalignments manually or use the IBM Content Manager V8.3 Fix Pack 1 version of the Java viewer toolkit.

Symptoms

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.

Resolving the problem

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 IBM Content Manager V8.3 Fix Pack 1 version of the Java viewer toolkit.

Receiving MalformedInputException message when using UTF-8 encoding

To resolve an exception message that you might get when you are using UTF-8 encoding, check the environment variables that are locale-specific. If the environment variables end with `UTF-8` suffix, remove the `UTF-8` suffix.

Symptoms

If your system locale is using a UTF-8 encoding, you might get `sun.io.MalformedInputException` message when using some SDK tools.

Resolving the problem

To find out whether your system is using a UTF-8 encoding, examine the locale-specific environment variables, such as `LANG` or `LC_ALL` attributes to see if they end with the suffix `UTF-8`.

To resolve the `sun.io.MalformedInputException` message, change the characters that are not within the 7-bit ASCII range (0x00 - 0x7f) and are not represented as Java Unicode character literals to Java Unicode character literals, for example: `\u0080`.

You can also work around this problem by removing the `UTF-8` suffix from the locale-specific environment variables.

For example, if your machine has default locale of `en_US.UTF-8`, set `LANG` to `en_US`.

Getting poor performance when you set WSE 3.0 diagnostics options on the WSE Settings 3.0 window

If you get poor performance when you set WSE 3.0 diagnostics options on the WSE Settings 3.0 window, remove the trace files completely to reduce the size of the files or disable the Message Trace option.

Symptoms

IBM Content Manager Web services supports Microsoft .NET Framework 2.0 and Microsoft WSE (Web Services Enhancement) 3.0. When you develop a Web services client application by using Microsoft Visual Studio 2005, you can set WSE 3.0 diagnostics options on the WSE Settings 3.0 window. However, your performance might decrease after you turn on the **Message Trace** option on the WSE Settings

3.0 window. Running the client application might cause a 100% usage of processing resources and also cause the memory usage to exceed 400® MB.

Causes

The WSE 3.0 settings affect the whole Visual Studio 2005 project, regardless of whether the build configuration is Debug or Release. After the **Message Trace** option is enabled, two trace files are generated to log all the activities. By default, these files are called `InputTrace.webinfo` and `OutputTrace.webinfo`. The size of these two files increases to hundreds of megabytes after a period of repeatedly running your client application that results in a performance degradation.

Resolving the problem

To solve this problem, you can perform either of the following procedures:

Procedure 1

Remove the trace files after running the application to ensure the size of the files is less than 50 MB.

Procedure 2

You can disable the **Message Trace** option on the **WSE Settings 3.0** window or edit the `app.config` file manually when you prepare to release your application.

Disable the Message Trace option on the WSE Settings 3.0 window

To disable the **Message Trace** option on the **WSE Settings 3.0** window:

1. From the Solution Explorer window in Visual Studio, right-click your project and open the **WSE Settings 3.0** window by clicking **WSE Settings 3.0** on the menu.
2. Clear the **Enable Message Trace** check box.
3.

Edit the `app.config` file manually

Open the `app.config` file from the project directory and set the value of the trace attribute enabled to false as in the following example.

```
<diagonostics>
<trace enabled="false" input="InputTrace.webinfo"
output="OutputTrace.webinfo" />
<detailedErrors enabled="true"/>
</diagonostics>
```

Notices

Trademarks

Index

Product Number: 5724-B19
5697-H60

SC27-1347-12