

IBM Content Manager for iSeries



Application Programming Guide and Reference

Version 5 Release 3

IBM Content Manager for iSeries



Application Programming Guide and Reference

Version 5 Release 3

Note

Before using this information and the product it supports, read the information in "Notices" on page 303.

Second Edition (May 2004)

This edition applies to Version 5 Release 3 of IBM Content Manager for iSeries (product number 5722-VI1) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SC27-1139-00

© Copyright International Business Machines Corporation 1997, 2004. All rights reserved.

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

Contents

About This Book vii

Who Should Use This Book	vii
How This Book Is Organized	vii
What's New in Version 5.3	viii
How to Use This Book	viii
Style Conventions	ix
Prerequisite and related information	ix
Support available on the Web.	x
iSeries Navigator	x
How to send your comments	x

Chapter 1. Introducing Content Manager for iSeries 1

A Closer Look at Content Manager for iSeries	1
Client/Server Relationship.	1
Content Manager for iSeries Components.	2

Chapter 2. Content Manager for iSeries Concepts 5

Understanding the Logical Data Model	5
Understanding Workflow	5
Getting Information about Documents and Folders	7
Supporting Case-Sensitivity	8
Naming Folders	8
Changing an Item's Index Class	8
Restricting Access to Items.	9
Migrating Objects.	9

Chapter 3. Application Programming Interfaces 11

Compiling and Linking Content Manager for iSeries Applications	11
Application Programming Interfaces	12
SimLibAddFolderItem (Add an Item to a Folder)	12
SimLibCatalogObject (Catalog an Object)	15
SimLibChangeIndexClass (Change the Index Class for an Item)	19
SimLibChangeObjectSMS (Change the SMS Criteria for an Object)	21
SimLibCloseAttr (Close an Attribute Set)	22
SimLibCloseObject (Close an Object)	23
SimLibCopyObject (Copy an Object)	25
SimLibCreateItem (Create an Item)	26
SimLibCreateObject (Create an Object)	29
SimLibDeleteItem (Delete an Item)	34
SimLibDeleteObject (Delete an Object)	36
SimLibFree (Free Memory)	37
SimLibGetAttrInfo (Get Attribute Information)	38
SimLibGetClassInfo (Get Index Class Information)	40
SimLibGetItemAffiliatedTOC (Get a Table of Contents for Item Affiliates)	41
SimLibGetItemInfo (Get Item Information)	43

SimLibGetItemSnapshot (Get a Snapshot of Item Attributes)	44
SimLibGetItemType (Get the Type of an Item)	46
SimLibGetItemXREF (Get a Cross-Reference for an Item)	47
SimLibGetSessionType (Get the Session Type)	49
SimLibGetTOC (Get a Table of Contents)	49
SimLibGetTOCData (Get a Snapshot of Attributes for a Group of Items)	53
SimLibListClasses (List Index Classes)	55
SimLibLogoff (Log Off)	56
SimLibLogon (Log On)	58
SimLibOpenItemAttr (Open Item Attributes)	61
SimLibOpenObject (Open an Object)	63
SimLibOpenObjectByUniqueName (Open an Object By its Unique Name)	66
SimLibQueryObject (Query an Object)	68
SimLibReadAttr (Read an Attribute)	69
SimLibReadObject (Read an Object)	70
SimLibRemoveFolderItem (Remove an Item from a Folder)	72
SimLibResizeObject (Resize an Object)	73
SimLibSaveAttr (Save an Attribute)	75
SimLibSearch (Search)	76
SimLibSeekObject (Seek an Object)	79
SimLibStageObject (Stage an Object)	80
SimLibStoreNewObject (Store a New Object in an Existing Item)	81
SimLibWriteAttr (Write an Attribute)	84
SimLibWriteObject (Write an Object)	85
SimWmActivateWorkPackage (Activate a Work Package)	87
SimWmBeginProcess (Start a Work Package on a Pre-defined Process)	88
SimWmChangeVariables (Change Variable Values for a Work Package)	90
SimWmCreateWorkPackage (Create a Work Package)	92
SimWmEndCollectionPoint (Force a Work Package Out of a Collection Point).	93
SimWmEndProcess (End a Work Package on a Process)	94
SimWmGetActionListInfo (Get Action List Information)	95
SimWmGetProcessInfo (Get Information About a Process)	96
SimWmGetWorkBasketInfo (Get Information about a Workbasket)	98
SimWmGetWorkPackage (Get the Next Work Package from a Workbasket).	99
SimWmGetWorkPackagePriority (Get the Priority of a Work Package)	101
SimWmListHistory (List the History of a Work Package)	102
SimWmListProcesses (List the Processes)	103
SimWmListWorkBaskets (List the Workbaskets)	104

SimWmMatchEvent (Satisfy an Event for a Work Package)	105
SimWmQueryVariables (Query Variables for a Specific Work Package)	107
SimWmQueryWorkPackage (Query a Work Package)	108
SimWmReturnWorkPackage (Return a Work Package to a Workbasket)	109
SimWmRouteWorkPackage (Route a Work Package)	111
SimWmSetWorkPackagePriority (Set the Priority of a Work Package)	112
SimWmSuspendWorkPackage (Suspend a Work Package)	114
Sim400ConvertCodepage (Code Page Conversion)	115
Sim400SendReceive (Send Data to AS/400)	116
Ip2CloseTOC (Close a Table of Contents)	117
Ip2GetLibSessionInfo (Get the Information for a Library Session)	118
Ip2GetTOCUpdates (Get the Updates to a Table of Contents)	119
Ip2ListAttrs (List the User-Defined Attributes)	121
Ip2ListContentClasses (List the Content Classes)	122
Ip2ListServers (List the Accessible Servers)	123
Ip2QueryClassPriv (Query the Privilege String for an Index Class or View)	124
Ip2QueryPrivBuffer (Query a Privilege Buffer)	125
Ip2TOCCount (Count the Items in a Table of Contents)	130
Ip2TOCStatus (Get the Status of a Table of Contents)	131

Chapter 4. Common Data Structures 133

Data Structures	133
AFFTOCENTRYSTRUCT (Affiliated Table of Contents Entry Structure)	133
ANNOTATIONSTRUCT (Annotation Information Structure)	134
ATTRINFOSTRUCT (Attribute Information Structure)	135
ATTRLISTSTRUCT (Attribute List Data Structure)	137
CLASSATTRSTRUCT (Class Attribute Structure)	138
CLASSINDEXATTRSTRUCT (Class Index Attribute Structure)	139
CLASSINDEXSTRUCT (Class Index Structure)	140
CLASSINFOSTRUCT (Index Class Information Structure)	141
CONTENTCLASSINFO (Content Class Information Structure)	142
HOBJ (Handle to Query Stored Object)	143
ICVIEWSTRUCT (Index Class View Information Structure)	143
ITEMINFOSTRUCT (Item Information Structure)	144
ITEMNAMESTRUCT (Item Name Data Structure)	146
LIBSEARCHCRITERIASTRUCT (Search Criteria Information Structure)	147
LIBSESSIONINFOSTRUCT (Library Session Information Structure)	148

NAMESTRUCT (Name Data Structure)	149
OBJINFOSTRUCT (Object Information Structure)	149
RCSTRUCT (Return Code Information Structure)	151
SERVERINFOSTRUCT (Server Information Structure)	153
SMS (System-Managed Storage Pointer)	154
SNAPSHOTSTRUCT (Snapshot Information Structure)	155
TOCENTRYSTRUCT (Table of Contents Entry Data Structure)	157
USERACCESSSTRUCT (User Access Data Structure)	158
USERLOGONINFOSTRUCT (User Logon Information Structure)	159
WMACTIONLISTFUNCSTRUCT (Action List Function Structure)	160
WMACTIONLISTINFOSTRUCT (Action List Data Structure)	161
WMHISTLOGENTRYSTRUCT (WMEvent History Structure)	162
WMLOCATIONINFOSTRUCT (Work Process Location Information Structure)	162
WMPROCESSINFOSTRUCT (Process Information Data Structure)	163
WMSNAPSHOTSTRUCT (Work Management Information Structure)	164
WMSUSPENDSTRUCT (Suspend Work Package Data Structure)	166
WMVARSTRUCT (Work Package Variable Data Structure)	167
WORKBASKETINFOSTRUCT (Workbasket Information Data Structure)	168

Chapter 5. Using the OLE Automation Interface 173

Programming with OLE Automation	173
Properties	173
Methods	173
Client for Windows Objects	173
Application Object	174
Documents Collection	174
Document Object	174
Error Object	174
Image Object	174
Items Collection	175
Item Object	175
Programming Tips	175
Releasing Objects	175
Handling Errors	175
Property and Argument Types	176
Sample Visual Basic Program	176
Properties and Methods of OLE Objects for Windows	177
Application Object	177
Document Object	186
Documents Object	189
Error Object	191
Image Object	192
Item Object	194

Items Collection	201
----------------------------	-----

Chapter 6. Sample High-Level Programming Interface 203

Sample High-Level Programming Interface for Visual Basic	203
General Use	203
Visual Basic Parameters and Variables	203
Access to the Client for Windows	204
Using Logon/Logoff with the Client for Windows	204
Samples of High Level Programming Interface APIs for Windows	205
VbVhlAddFolderItem (Add an Item to a Folder)	205
VbVhlAdminItemNoteLog (Administer Document Note Logs)	206
VbVhlChangeItemIndex (Change an Item's Index Class)	207
VbVhlCloseDocViews (Close the Document Image View Window)	209
VbVhlCopyDoc (Create a Copy Of a Document)	210
VbVhlCreateFolder (Create a New Folder)	212
VbVhlCreateFolderAddItem (Create a Folder and Add an Item)	213
VbVhlDeleteItem (Delete an Item)	215
VbVhlDisplayDocView (Display a Document Image)	216
VbVhlDisplayVhlItem (Display Item Using the Client for Windows)	217
VbVhlDropFuncs (End Access to VHLPI Functions)	218
VbVhlExportDocObj (Export a Document Base Object)	219
VbVhlGetVlUserID (Get the Logon User ID)	220
VbVhlImportDocObj (Import a Document Base Object)	220
VbVhlListContClasses (List all Content Classes)	222
VbVhlListFolderItems (List Folder Contents)	223
VbVhlListFolderItemsAttr (List Folder Contents and Their Attributes)	225
VbVhlListIndexClassAttr (List All Attributes Of an Index Class)	227
VbVhlListIndexClasses (List all Index Classes)	229
VbVhlListItemCC (List a Base Object's Content Class)	230
VbVhlListItemInfo (List an Item's Index Class and Attribute Information)	231
VbVhlListWBItems (List Workbasket Contents)	233
VbVhlListWorkBaskets (List All Workbasket Names)	234
VbVhlLoadFuncs (Get Access to VHLPI Functions)	235
VbVhlLogoff (End Access to IBM Content Manager for iSeries)	236
VbVhlLogon (Get Access to IBM Content Manager for iSeries)	237
VbVhlRemoveFolderItem (Remove an Item From a Folder)	238
VbVhlScanDoc (Scan Documents)	239
VbVhlSearchAdv (Advanced Search for Items)	239
VbVhlSearchItem (Search for Items)	241

Chapter 7. Content Manager for iSeries Programming Interface APIs on the Server 245

Server Versions of the Content Manager for iSeries Client APIs	245
Server-only Content Manager for iSeries APIs	245
QVISNDRCV (Send and Receive Buffer)	245

Chapter 8. Content Manager for iSeries User Exits 249

Client User Exits	249
AlternateSearchUserExit (alternate search user exit)	249
ChangeSMSUserExit (change system-managed storage user exit)	251
DetNextWBUserExit (determine next workbasket user exit)	254
DetermineWorkflowUserExit (determine workflow user exit)	258
GetAttributeValueList (Get attribute value list)	262
GetValueListLength (Get value list length)	263
OverloadTriggerUserExit (overload trigger user exit)	264
QuerySortUserExit (query sort user exit)	268
SaveRecordUserExit (save record user exit)	272
UserActionUserExit (Workflow User Action User Exit)	276
UserOptionUserExit (User-option User Exit)	277
WBItemSelectedUserExit (Workbasket Item Selected User Exit)	277
WBItemCompletedUserExit (Workbasket Item Completed User Exit)	278
UserDefinedWBUserExit (User-defined Workbasket User Exit)	279
Server User Exits	280
Logon User Exit	281
Logoff User Exit	281
Save Attributes User Exit	281
Create Object User Exit	282
Delete Object User Exit	283
Open Object User Exit	283
Create Item User Exit	284
Item Created User Exit	284
Delete Item User Exit	285
Object Import Create Item User Exit	285
Object Import Item Created User Exit	286
Add Folder Item User Exit	286
Route Work Package User Exit	287
Get Work Package User Exit	287
Return Work Package User Exit	288
End Process User Exit	289
Set Variable User Exit	289
Server User Exit for Process Definitions	290

Appendix A. Guidelines for Search Expressions 291

Logical Operators for Searches	291
Search Expressions	291
Attribute	291
Operator	291

Value	292
Relational Operators for Searches.	292
Process/Location Search.	293

Appendix B. Predefined Content Classes 295

Appendix C. External References . . . 299
Creating External References 300

Notices 303
Trademarks 305

Glossary 307

Index 315

About This Book

This book describes how to create or integrate image, workflow, or other applications into a Content Manager for iSeries system. These application programming interfaces (APIs) support client application development for Content Manager for iSeries. The information in this book applies to application development in a 32-bit Windows® programming environment.

This book explains the following:

- How to use the various components of Content Manager for iSeries.
- Tips for identifying application requirements as you create a Content Manager for iSeries application.
- Ways to use the APIs to write image, workflow, or other applications that use Content Manager for iSeries APIs.
- The terminology used with Content Manager for iSeries.

Who Should Use This Book

If you are an application programmer responsible for developing image, workflow, or other applications, this book provides detailed information about each function available to you through the APIs.

If you are a systems designer or integrator who is designing a Content Manager for iSeries system or application, you need to understand how Content Manager for iSeries works and how to create new applications for, or integrate existing applications with, Content Manager for iSeries. This book describes how each component and its corresponding functions can meet your technical, design, and business requirements for imaging, workflow, or other applications.

If you are a system administrator responsible for administering and supporting Content Manager for iSeries implementations, you can use this book as a reference.

To successfully program with Content Manager for iSeries, you need experience developing applications in C, COBOL, or RPG and the OS/400® environment for server-side programming. For client-side programming, you need experience with OLE, VisualBasic, C++ and/or C, as well as experience with the Windows environment.

How This Book Is Organized

This book contains the following information.

- Chapter 1, "Introducing Content Manager for iSeries," on page 1 introduces the software and hardware components of Content Manager for iSeries and the APIs available with Content Manager for iSeries.
- Chapter 2, "Content Manager for iSeries Concepts," on page 5 introduces you to Content Manager for iSeries concepts and capabilities.
- Chapter 5, "Using the OLE Automation Interface," on page 173 shows you how to enable another Windows-based application to log on to Content Manager for iSeries and perform various tasks within the Client for Windows using APIs that are based on OLE 2.0 Automation.

- “Sample High-Level Programming Interface for Visual Basic” on page 203 shows you how to enable another Windows-based application to log on to Content Manager for iSeries and perform various tasks within the Client for Windows using APIs that are based on OLE 2.0 Automation.
- Chapter 3, “Application Programming Interfaces,” on page 11 describes the Content Manager for iSeries common application programming interfaces.
- Chapter 4, “Common Data Structures,” on page 133 describes the common data structures and database tables you can use to manipulate and manage objects and classes of objects.
- “Properties and Methods of OLE Objects for Windows” on page 177 describes the properties and methods associated with all client application objects.
- Chapter 6, “Sample High-Level Programming Interface,” on page 203 provides samples of high level application programming interfaces for windows.
- Chapter 7, “Content Manager for iSeries Programming Interface APIs on the Server,” on page 245 provides information about the Content Manager for iSeries server versions of APIs.
- Chapter 8, “Content Manager for iSeries User Exits,” on page 249 gives you the Content Manager for iSeries user exits.
- Appendix A, “Guidelines for Search Expressions,” on page 291 gives you some guidelines to follow when you are searching the Client for Windows .
- Appendix B, “Predefined Content Classes,” on page 295 lists the predefined content classes for Content Manager for iSeries.
- Appendix C, “External References,” on page 299 describes how to access data in other repositories by using the Content Manager for iSeries Windows client and programming interfaces.

What’s New in Version 5.3

This edition of IBM® *Content Manager OnDemand for iSeries™: Application Programming Guide and Reference* contains new technical information. There may be some instances where changes were made, but change bars are missing. Significant changes to note are:

Expanded the capability to store ten-character userids. In previous releases, only the first eight characters of the userid were used. **Important:** Many files have been modified to support ten-character userids. If you support external references and read or write to the EKD0314 file, it might be necessary to recompile your custom programs to support the expansion of the userid field in the file format.

How to Use This Book

Use Chapter 1, “Introducing Content Manager for iSeries,” on page 1 to familiarize yourself with Content Manager for iSeries. Refer to Chapter 2, “Content Manager for iSeries Concepts,” on page 5 for conceptual information about how to use the Content Manager for iSeries components.

Style Conventions

To help you understand the text, this book uses the following conventions:

Convention	Stands for
Upper and lowercase	Column names in library server database Tables (example: Owner UserID)
UPPERCASE	Column names in object server database tables Constants Data structure names Data types Database table names Return codes from function calls
Bold Mixed Case	API function names (example: SimLibLogon)
BOLD UPPERCASE	Field values to specify Parameter values to specify
<i>ITALIC UPPERCASE</i>	The maximum length of a field
<i>Italic</i>	Field names in data structures Names of books as references Parameter names in API functions Terms defined for the first time in the book

Prerequisite and related information

Use the iSeries Information Center as your starting point for looking up iSeries technical information. You can access the Information Center in one of two ways:

- From the following Web site: <http://www.ibm.com/eserver/iseries/infocenter>
- From CD-ROMs that ship with your Content Manager for iSeries order:
iSeries Information Center, SK3T-4091-04. This package also includes the PDF versions of the Content Manager for iSeries publications in *iSeries Information Center: Supplemental Manuals*, SK3T-4092-01, which replaces the Softcopy Library CD-ROM.

The IBM iSeries Information Center contains advisors and important topics such as CL commands, system application programming interfaces (APIs), logical partitions, clustering, Java™, TCP/IP, Web serving, and secured networks. It also includes links to related IBM Redbooks™ and Internet links to other IBM Web sites such as the Technical Studio and the IBM home page.

Go to <http://www-3.ibm.com/software/data/cm/cmgr/400/library.html> to access the Content Manager for iSeries publications from the product Web site. The publications are listed in Table 1.

Table 1. IBM Content Manager for iSeries 5.3 publications

Title	Publication number
<i>IBM Content Manager for iSeries: Planning and Installing</i>	SC27-1133
<i>IBM Content Manager for iSeries: Getting Started with Client for Windows</i>	GC27-1135

Table 1. IBM Content Manager for iSeries 5.3 publications (continued)

Title	Publication number
<i>IBM Content Manager for iSeries: System Administration Guide</i>	SC27-1136
<i>IBM Content Manager for iSeries: Messages and Code</i>	SC27-1137
<i>IBM Content Manager for iSeries: Understanding Advanced Workflow</i>	SC27-1138
<i>IBM Content Manager for iSeries: Application Programming Guide and Reference</i>	SC27-1139

Support available on the Web

Product support is available from IBM support at <http://www-3.ibm.com/software/data/cm/cmgr/400/support.html>.

iSeries Navigator

IBM iSeries Navigator is a powerful graphical interface for managing your iSeries servers. iSeries Navigator functionality includes system navigation, configuration, planning capabilities and online help to guide you through your tasks. iSeries Navigator operation and administration of the server easier and more productive and is the only user interface to the new advanced features of the OS/400 operating system. It also includes Management Central for managing multiple servers from a central server.

For more information about iSeries Navigator, see the Information Center.

How to send your comments

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

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

Chapter 1. Introducing Content Manager for iSeries

This overview explains the ways to implement Content Manager for iSeries components. This information is a framework for you to use to determine how to make the most of the Content Manager for iSeries APIs as you create your applications. It includes an overview of the following Content Manager for iSeries components:

Client Application Program

The client application you use can be the client application program delivered with Content Manager for iSeries or an application that you develop.

Content Manager for iSeries APIs

Content Manager for iSeries APIs are high-level programming interfaces that let you access and manipulate data stored on a host server.

Client Interfaces for Windows

The client APIs for Windows provide a programming interface you can use to develop your own Windows-based client applications for Content Manager for iSeries.

With Content Manager for iSeries, you can develop a customized document management solution that includes a host server and information-processing capabilities for multiple media types. Using Content Manager for iSeries, you can create image and other applications to automate and gain control of the information your enterprise processes each day. You can increase productivity and security, lower storage costs, and improve customer service.

Content Manager for iSeries offers tailorable document processing for both large and small organizations. Content Manager for iSeries lets users capture, store, and retrieve documents on-line and provides document, folder, and work management capabilities. Content Manager for iSeries also provides extensive data integrity and security.

Content Manager for iSeries consists of Windows clients connected to an iSeries server. It provides enterprise-wide access to document processing, storage, and management. That way, Content Manager for iSeries lets multiple departments of an enterprise, located in one or several locations, access their own documents as well as enterprise documents.

A Closer Look at Content Manager for iSeries

Content Manager for iSeries offers a complete document management system through its client/server architecture. Once you understand the client/server concept, you can then take a closer look at all the key components that make up Content Manager for iSeries.

Client/Server Relationship

Content Manager for iSeries consists of a client connected to one or more host servers. The host server maintains document and folder index information, document and folder relationships, work-in-process information, and interacts with the client.

Content Manager for iSeries Components

Content Manager for iSeries consists of a client, the client application program, a host server, and Content Manager for iSeries APIs. You can use Content Manager for iSeries to develop additional clients.

The following figure shows the major components of Content Manager for iSeries.

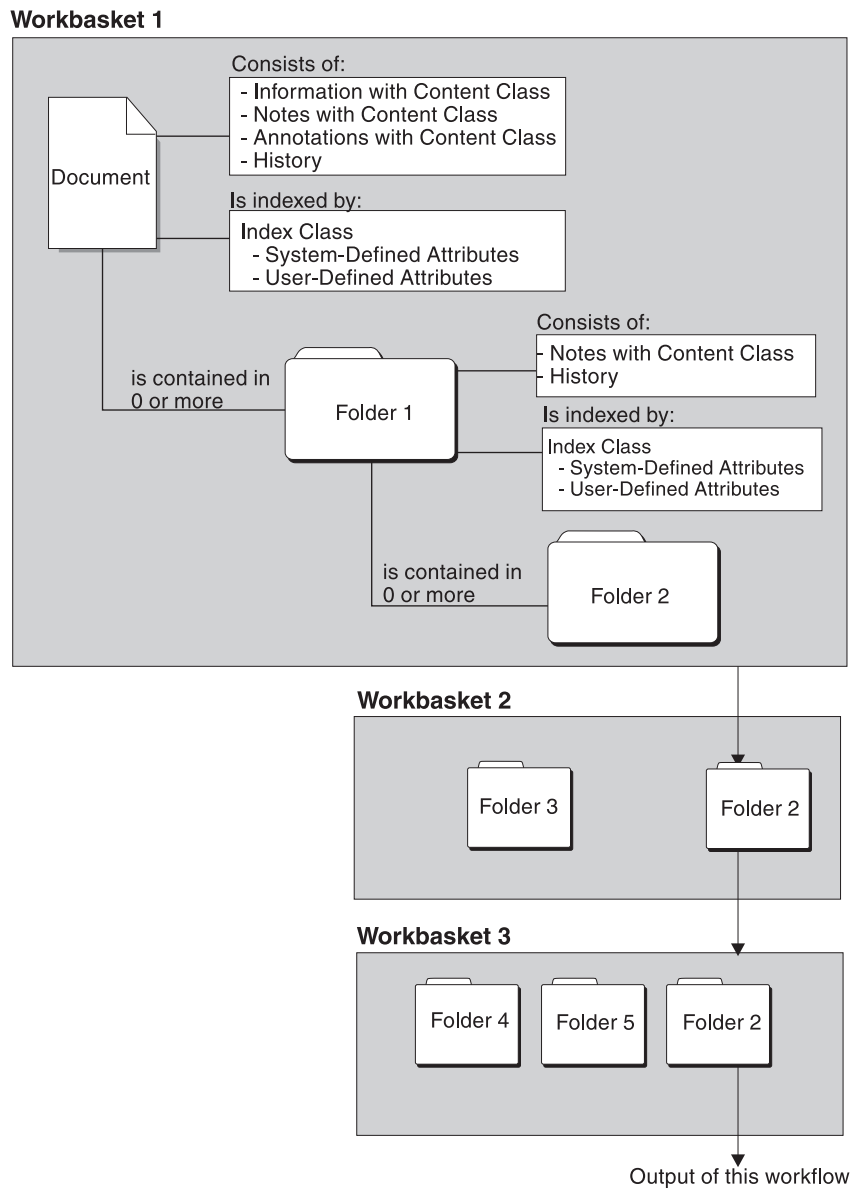


Figure 1. The Main Components of Content Manager for iSeries

Client Application

The Content Manager for iSeries client application provides document and folder management, scanning support, import and export, work management, and search capabilities built on the Content Manager for iSeries APIs.

The client application program provides a complete end-user interface for Content Manager for iSeries. You can configure the client application program to meet the

specific needs of your enterprise. User exits provide points where you can provide application-specific processing routines to customize the client application program.

The client application program provides APIs to let you integrate folder management, work management, and document management with your existing information systems. You can easily integrate your custom software and other applications with the client application program.

You can use the client application program that comes with Content Manager for iSeries, write your own application, or use an application available from IBM Services or Business Partners.

Content Manager for iSeries APIs

If you choose to write your own application, you can use the Content Manager for iSeries APIs as the primary interface between the Content Manager for iSeries host server and your application.

In the Content Manager for iSeries data model, the most basic components are documents, folders, *workbaskets*, and work packages. *Documents* are similar to paper documents. *Folders* are similar to folders in a paper filing system and can contain other folders or documents. A *workbasket* is a queue of work for one or more employees to use. It is similar to an in-basket from which to take work. A *work package* is an entry in a workbasket for use in work management and contains a document or folder.

Depending on the level of access to documents, you can perform the following operations using these APIs:

- Store a document
- Index a document or folder
- Retrieve a document or folder

The APIs support a wide range of the functions available in Content Manager for iSeries. You can use these APIs to create Windows or OS/400 applications.

Content Manager for iSeries Server

The Content Manager for iSeries server uses IBM's relational database technology to maintain document contents and provides data integrity by performing the following functions:

- Manage data
- Maintain index information
- Control access to documents stored in object servers

You can develop applications to reference multiple Content Manager for iSeries servers.

Chapter 2. Content Manager for iSeries Concepts

This section provides an overview of the Content Manager for iSeries concepts, including the logical data model. In other products of the IBM Content Manager for iSeries family, the term “folder manager data model” identifies a subset of application programming interfaces (APIs) and “common application programming interface” (CAPI) identifies a subset of *SimLib* interfaces. In Content Manager for iSeries, all available programming interfaces are known as Content Manager for iSeries APIs.

Understanding the Logical Data Model

Content Manager for iSeries implements the folder manager data model, which includes concepts such as items, objects, folders, index classes, and attributes. This model provides your application with many capabilities for managing business objects. Documents in Content Manager for iSeries are similar to paper documents. A document consists of a set of closely related objects, such as pages in a letter or report. Documents can contain one or more parts. These parts, known as base parts, can be pages or illustrations in a letter, report, or other documents. Other parts associated with documents are annotations and notes.

An annotation part associated with a document can highlight sections of a document. A note part associated with a document is textual information that you attach to the document to give additional information to other users. For example, you might attach a note to draw the reader’s attention to part of the document. An event part associated with a document provides a historical trail of the processing you perform on the document.

Folders in Content Manager for iSeries are similar to folders in a paper filing system. Each folder can contain one or more documents or other folders. Each folder has a table of contents that lists all the documents and folders it contains. You can associate note parts with a folder.

Understanding Workflow

Workflow describes the movement and processing of work. The terms *workflow* and *work management* are used interchangeably. Workflow is the definitions and rules that govern how work is performed.

The following terms are commonly used in descriptions of workflow:

Action list An approved list of the actions, defined by a supervisor, that a user can perform on work packages.

Ad hoc process A process that is not a defined workflow process. An *ad hoc process* is started when a user creates a work package and assigns it directly to a workbasket. The user manually routes the work package from one workbasket to another by reassigning it. Within workflow processing, the value *ADHOC is used in place of process names to indicate that the work package is being routed in an ad hoc manner.

Collection point

The point where work packages wait for specific events to either occur or become synchronized before processing can continue.

A collection point is part of a process. For example, a collection point is where work packages that are part of the process "open a new account" must wait until credit information is verified.

Decision point

The point where work packages continue on their current route or switch to an alternate route, depending on the specific information in each work package. Decision points are tables consisting of variable names, values, and routes.

A decision point is part of a process. For example, a decision point is where work packages that are part of the process "open a new account" receive approval or not based on credit information.

Instance

An occurrence of a work package within a process. If the process consists of parallel routes, multiple instances of a work package exist.

Process

The series of steps, events, and rules through which a work package flows. A process is a combination of the route, collection point, and decision point through which a predefined type or work package must progress.

For example, a process called "open new account" would describe the following:

- The steps that work packages related to opening a new account must follow
- The events (such as verifying credit information) that must occur before work packages for new accounts can be routed to another point in the system
- The decisions that determine whether to open a new account based on the information for that particular account (for example, a good credit rating versus a poor one).

Suspend

To hold a work package at a workbasket until stated criteria have been satisfied. Work packages can be suspended for multiple criteria, therefore multiple suspend requests can exist for a work package. A document work package can be suspended for a specific date. A folder work package can be suspended for a specific date or index class.

A suspended work package is released when the criteria have been met, or when a user with proper authority overrides the criteria and manually releases pending requests.

Work package

The work that is routed from one location to another. A work package can consist of a document, a folder, or a customer-defined collection of objects. Work packages can be routed automatically by defined processes, or users can manually route work packages in an ad hoc manner to workbaskets they specify. Users access and work with work packages through workbaskets.

Workbasket

A container that holds work packages. Workbaskets can be used as parts of process definitions and ad hoc routes. A workbasket definition includes the rules that govern the presentation, status, and security of its work packages.

Getting Information about Documents and Folders

To read the attributes of a document or folder, an application can open the item (**SimLibOpenItemAttr**), read one attribute at a time (**SimLibReadAttr**), and close the item (**SimLibCloseAttr**). You can also use **SimLibGetItemSnapshot** to retrieve all the attributes and optional information. This function retrieves the system attributes, user-defined attributes, workflow information, checkout holder, and other data about the folder or document. Use this function if you want all of this information and do not need to open the item for subsequent activities.

SimLibSearch can be used to retrieve user-defined attributes for items matching a predefined search criteria.

If the snapshot option flag includes system attributes (**SIM_SYSTEM_ATTR**), **SimLibGetItemSnapshot** returns four attributes in the **ATTRLISTSTRUCT** array for the current view in addition to user-defined attributes:

- **OIM_ID_ITEM_NAME**
- **OIM_ID_CREATE_TIMESTAMP**
- **OIM_ID_MODSYS_TIMESTAMP**
- **OIM_ID_UID**

Your application must not depend on the order of appearance of the attributes or on whether user-defined or system attributes come first.

Instead of **SimLibGetItemSnapshot**, use **SimLibGetTOCData** to return a snapshot for an entire list of items. The **TOCENTRYSTRUCT** array returned by **SimLibGetTOC** can be passed directly to **SimLibGetTOCData** for processing as a group, if its number of entries does not exceed **SIM_TOC_MAX_ENTRY_COUNT**. If the count exceeds the maximum, pass the entries, up to the maximum, one at a time. Then, advance to the next batch in the **TOCENTRYSTRUCT** array. The list pointer to **SimLibGetTOCData** can reference an entry in the array, and the function begins processing at this entry.

For example, your application can have basic logic similar to the following:

```
u1RC = SimLibGetTOC(hSession,...);
if (u1RC != SIM_RC_OK) {
    // process errors
} else {
    ulCount = count returned by SimLibGetTOC
    pTOC = TOCENTRYSTRUCT array pointer returned by SimLibGetTOC
    while (ulCount > 0) {
        i = minimum of ulCount and SIM_MAX_TOC_ENTRY_COUNT
        u1RC = SimLibGetTOCData(hSession,pTOC,i,NULL,pRC);
        if (u1RC != SIM_RC_OK) {
            // process errors, possibly exit the loop
        } else {
            // process results
            call SimLibFree to release data returned
        }
        ulCount -= i; // decrement number left to do
        pTOC += i;   // advance to next set, if any
    }
    close the TOC from SimLibGetTOC
}
```

When you are logged on, you must have sufficient privileges to get the attributes for each item, or the **SimLibGetTOC** function returns an error.

You still might want to take advantage of the efficiency of **SimLibGetTOCData**, without processing the entire set of items from **SimLibGetTOC**. **SimLibGetTOCData** skips an item ID in the TOCENTRYSTRUCT that is a NULL string. Because an application might not modify the TOCENTRYSTRUCT array returned by the **SimLibGetTOC** function, copy the TOCENTRYSTRUCT array to another buffer, and then set the item ID to NULL. You can also filter the unnecessary entries by copying the desired data to a temporary TOCENTRYSTRUCT array and passing that to **SimLibGetTOCData**. If the item ID is NULL, **SimLibGetTOCData** still returns an empty SNAPSHOTSTRUCT for the item.

You can use the same approach for processing a block of items even when they are not returned by **SimLibGetTOC**. Your application can generate its own list in the same format and pass that list into **SimLibGetTOCData**. As an example, you can take the results of a search (**SimLibSearch**) and build the TOCENTRYSTRUCT array from the item ID list. **SimLibGetTOCData** requires the index class of each item in advance. **SimLibSearch** does not return the index class, but if you restrict the search to a single index class, your application already knows the index class of each item returned by the search.

You can also use **SimLibSearch** directly to retrieve user-defined or both user-defined and system-defined attributes by using the SIM_SEARCH_USER_ATTR or the SIM_SEARCH_USER_SYSTEM_ATTR option. This is more efficient than calling **SimLibSearch** to get the item IDs, and then calling other APIs, such as **SimLibGetTOCData**, to retrieve attribute information.

Even though you make a TOCENTRYSTRUCT array that might look like the array from **SimLibGetTOC**, you cannot use a table of contents function such as **Ip2TOCUpdates** on a simulated TOC. Table of contents functions require a handle returned by **SimLibGetTOC**.

Supporting Case-Sensitivity

| Content Manager for iSeries stores character-string attributes exactly as presented
| by the application. Content Manager for iSeries always converts user IDs to
| uppercase.

Naming Folders

The folder data model for Content Manager for iSeries does not include a folder name. A folder name such as a customer name, customer number, case name, or other recognizable text is an index class attribute for a class that uses a folder name. To search for a folder by name, therefore, your application must know the relevant index classes with folder names and construct the appropriate search.

Changing an Item's Index Class

When you create an item, it is associated with an index class. When your application changes the index class of the item, this entry is updated to reflect the change. This entry always contains the current index class to which the item belongs. A number of Content Manager for iSeries APIs, including **SimLibGetItemInfo** and **SimLibGetItemSnapshot** return this information to your application. You should use this index class within your application.

Restricting Access to Items

There are two layers for access control: the privileges that are defined for a user and access lists. The user privileges are often referred to as general privileges. Access lists are used to establish access to index classes, workbaskets and processes. An access list is a combination of a list of users and a set of privileges. Access lists **add** authority to general privileges; they do not remove authority.

In the simplest example of authority control, all users have access to all items in the library. To implement this type of authority control, give all users maximum privileges. Since access lists add authority, it is not necessary in this example to implement any access lists for your index classes, workbaskets or processes. There are, however, many available levels of restricted access.

One type of restriction is to allow a subset of users to have access to specific folders and documents. To do this, you would first define general privileges for all users specifying minimum access to the index class for the items. You would then define a list consisting of those users and groups that are allowed to work with the index class. That list of users is then associated with privilege sets that allow index class functions.

The list of users combined with the special privilege settings produces an access list that is then used for the index class. In this way, users that are not part of the access list are denied use of the index class and users that are part of the access list are allowed to perform those functions specified in the privilege set.

SimLibLogon returns general privileges. **Ip2QueryClassPriv** returns privileges for index classes. Similarly, **SimWmGetWorkBasketInfo** and **SimWmGetProcessInfo** return privileges for the workbasket or process. Your application can use these privilege strings to establish in advance whether to offer specific functional options to users. For example, your application can let a user view an item for which the user does not have delete authority without offering the delete option.

Migrating Objects

The Content Manager for iSeries storage management function allows objects to be moved from one medium to another—from magnetic disk to optical storage, for example—based on controls that the administrator establishes. A collection name is assigned to each object created in the system. A collection defines the storage management controls associated to a group of objects that typically have similar performance, availability, backup, and retention characteristics. An application can assign an object to a different collection using the **SimLibChangeObjectSMS** API.

Chapter 3. Application Programming Interfaces

This section describes the formats and parameters of the Content Manager for iSeries application programming interfaces (APIs). You can recognize these APIs by their **SimLib**, **SimWm**, **Sim400**, and **Ip2** prefixes.

For more information about the data structures for these APIs, see Chapter 4, "Common Data Structures," on page 133.

Compiling and Linking Content Manager for iSeries Applications

Content Manager for iSeries can be accessed through the Content Manager for iSeries APIs. You need the following files to build and run applications to access Content Manager for iSeries:

- EKDVIAPI.H** The structures, macros, and function prototypes for the Content Manager for iSeries APIs. EKDVIAPI.H includes the following header files:
 - EKDVIERR.H** Error numbers and descriptive names. The name is logged in Content Manager for iSeries for any error detected.
 - EKDVLIB.H** Library API definitions.
 - EKDVITYP.H** Constants and common type definitions.
 - EKDVIWM.H** Workflow API prototypes.
- EKDWS.LIB** LIB file required to link with EKDWS.DLL.
- EKDWS.DLL** All API functions.
- EKDWS35I.DLL**
IBM VisualAge runtime DLL.

These files are installed when you install the IBM Content Manager for iSeries Windows Client Toolkit.

Applications must access headers as follows:

```
#include "EKDVIAPI.H"
```

If you are not using VisualAge, the LIB file must be regenerated using ILIB or an equivalent command.

The Content Manager for iSeries APIs use code page conversion tables from VisualAge. Your installation program should install the required files for the code pages that are to be used for any given installation. The code page conversion files are located in the FRNROOT\ICONV and FRNROOT\UCONVTAB directories.

You must set the LOCPATH environment variable to the directory above (FRNROOT). You can do this in AUTOEXEC.BAT or the Registry, or your application can do it before the call to **SimLibLogon**. Doing this ensures that the variable is always set, which prevents conflicts with other products.

Client tracing and logging can be enabled to aid in problem determination. The environment variables below can be set to any value to control tracing. Results are

logged to VI400.LOG in the working directory or path specified in the VI400_LOG_PATH environment variable. The file is overwritten when the first call is made (such as to **SimLibLogon**).

VI400_LOG_PATH

Path for VI400.LOG

VI400_LOG_TRACE

Function entry and exit

VI400_LOG_PERFORMANCE

Trace and data transmission time

VI400_LOG_DATA

Data sent to and received from the iSeries system

VI400_LOG_STORAGE

Content Manager for iSeries object storage allocation and de-allocation

VI400_LOG_LOCKS

Log lock and unlock operations for each API

VI400_LOG_ALL

All trace levels

The FRNOLINT.TBL file is used to contain entries that define Content Manager for iSeries servers. It must be located in the path from which the program was started or the path contained in the VI400_CONFIG_PATH environment variable. The following is an APPC and a TCP/IP example:

```
SERVER: MYVI400 REMOTE APPC
        LU_NAME      = USIBMNR.AS400DS1
        TP           = EKDCS01P.EKDCS01P.QVI
        MODE         = QPCSUPP
        SERVER_TYPE  = FRNLS400

SERVER: MYVI400 REMOTE TCPIP
        HOSTNAME     AS400DS1
        PORT         31098
        SERVER_TYPE  = FRNLS400
```

In this example, if the database name passed to **SimLibLogon** is MYVI400, the above entry would be used to connect to the iSeries system. Since the path in the VI400_CONFIG_PATH environment variable accesses FRNOLINT.TBL, it can be placed on a network drive or in a directory on an iSeries that is accessed through Client Access or an equivalent product. If the environment variable is not set, the file is accessed in the current directory – namely, the **Start in** directory specified in the **Shortcut** page of the **Properties** for the icon.

EKDVIERR.H should be in the path defined in VI400_CONFIG_PATH. This file is used to log the descriptive name of each Content Manager for iSeries return code.

Application Programming Interfaces

SimLibAddFolderItem (Add an Item to a Folder)

<p>Format</p> <pre>SimLibAddFolderItem(<i>hSession, pszFolderID, pszItemID, pAsyncCtl, pRC</i>)</pre>
--

Purpose

Use the **SimLibAddFolderItem** function to add a document or a folder item to an existing folder.

Parameters

<i>hSession</i>	HSESSION — input
	The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszFolderID</i>	PITEMID — input
	The identifier of the folder. Use the item ID of an existing folder to which you want to add a document or a folder item. This folder does not need to be open.
<i>pszItemID</i>	PITEMID — input
	The identifier of an item. Use the item ID of the document or the folder item that you are adding to the folder. The item cannot already exist in the folder. Do not use the identifier of the same folder that you specified in the <i>pszFolderID</i> parameter. You cannot add a folder to itself.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input
	Not supported.
<i>pRC</i>	PRCSTRUCT — input/output
	The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in the RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field
<i>ulParam1</i>	The function does not use this field
<i>ulParam2</i>	The function does not use this field
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_ITEM_OR_FOLDER • SIM_RC_INVALID_PITEMIDFOLDER_PTR • SIM_RC_INVALID_PITEMIDFOLDER_VALUE • SIM_RC_INVALID_PITEMIDITEM_PTR • SIM_RC_INVALID_PITEMIDITEM_VALUE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_OUT_OF_MEMORY • SIM_RC_PITEMIDFOLDER_NOT_A_FOLDER • SIM_RC_PITEM_NOT_FOLDER_OR_DOCUMENT • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Preparation:

- To create a folder, use the **SimLibCreateItem** function.
- A document or folder can be in multiple folders at the same time.
- A folder and the items it contains can all have different index classes.

Restrictions:

- You cannot add a folder to itself.
- This function does not automatically update the temporary copy of the folder table of contents. You must use the **Ip2GetTOCUpdates** or **Ip2GetTOC** function to update your temporary copy of the folder table of contents.

Example

```
#include <windows.h>                /* Main Windows header files */
#include <sys\types.h>              /* Standard I/O header files */
#include <stdio.h>                  /* Standard library header files */
#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <io.h>
#include "ekdviapi.h"              /* Content Manager for iSeries */

main ()
{
    HSESSION    hSession;           /* Product session handle */
    PITEMID     pszFolderID;        /* ID of the folder */
    PITEMID     pszItemID;          /* ID of the item to be added */
    RCSTRUCT    RCStruct;           /* RC data structure */
    USHORT      sResult;            /* return codes */

    /******
    /*Initialize folderID and itemID
    /******
    memset (pszFolderID, '\0', DOC_ID_SIZE); /* set to null */
    strcpy ((CHAR *)pszFolderID, (CHAR *) "F000000001");

    memset (pszItemID, '\0', DOC_ID_SIZE); /* set to null */
    strcpy ((CHAR *)pszItemID, (CHAR *) "DA97220AA.AAB");

    /******
    /* Call SimLibAddFolderItem to place a new document in a folder */
    /******

    sResult = SimLibAddFolderItem(
        hSession,                    /* ses'n handle from SimLibLogon */
        pszFolderID,                 /* add item to this folder */
        pszItemID,                   /* add this item to above folder */
        (PASYNCCTLSTRUCT) NULL,      /* Request SYNCHRONOUS processing*/
        (PRCSTRUCT) &RCStruct        /* Pointer to RC data structure */
    );

    if (sResult != SIM_RC_OK) {
        printf("Add folder item failed \n");
    }
}
```

Related Functions

- **SimLibGetTOCData**
- **Ip2GetTOCUpdates**
- **Ip2TOCCount**
- **SimLibGetTOC**

- SimLibRemoveFolderItem

SimLibCatalogObject (Catalog an Object)

Format

```
SimLibCatalogObject( hSession, hObj, ulConCls, pSMS, pszFullFileName,
ulPriority, fCreateControl, ulVersion, lSeqAfterPart, ulAffiliatedType,
pAffiliatedData, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibCatalogObject** function to create a new object from the file that you specify. Use this function when your data is already in a file rather than in memory.

Your application can substitute this function for the following sequence of Content Manager for iSeries functions:

- SimLibCreateObject
- SimLibOpenObject
- SimLibWriteObject
- SimLibCloseObject

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObj</i>	HOBJ — input The pointer to an object handle block in the HOBJ data structure. For more information on the HOBJ data structure, see “HOBJ (Handle to Query Stored Object)” on page 143. “Guidelines for Use” describes the effects of your input to this data structure.
<i>ulConCls</i>	ULONG — input The content class identifier for the object (see Appendix B, “Predefined Content Classes,” on page 295). The value of this parameter tells what kind of data is in the object that you are cataloging. To indicate an undefined content class, specify the value SIM_CC_UNKNOWN for this parameter. However, if you do not use a defined content class, other applications cannot use Content Manager for iSeries content class services to determine how to manipulate the contents of objects that you store.
<i>pSMS</i>	PSMS — input Pointer to a system-managed storage (SMS) structure for an object. This structure uses only <i>szCollectionName</i> .
<i>pszFullFileName</i>	PSZ — input The pointer to a fully qualified directory path and file name
<i>ulPriority</i>	USHORT — input Not supported.

SimLibCatalogObject

<i>fCreateControl</i>	BITS — input
	Control option bits for the cataloging operation. The valid values are:
	SIM_CLOSE Closes the object on completion of the request.
	SIM_OPEN Leaves the object open in update mode.
<i>ulVersion</i>	ULONG — input
	Not supported.
<i>lSeqAfterPart</i>	LONG — input
	Not supported.
<i>ulAffiliatedType</i>	LONG — input
	The type of affiliated object. The defined values are:
	SIM_ANNOTATION Indicates that the object is an annotation associated with a folder or a document.
	SIM_BASE Indicates that the object is a base object such as a Mixed Object Document Content Architecture (MO:DCA) or Tag Image File Format (TIFF) file.
	SIM_EVENT Indicates that the object is an event associated with a folder or a document.
	SIM_MGDS Indicates that the object is an MGDS (machine-generated data stream) associated with a folder or a document.
	SIM_NOTE Indicates that the object is a note associated with a folder or a document.
<i>pAffiliatedData</i>	PVOID — input
	The pointer to a data structure of the type ANNOTATIONSTRUCT. If the <i>ulAffiliatedType</i> parameter contains the value SIM_ANNOTATION, <i>pAffiliatedData</i> points to this structure, which contains additional data affiliated with the object. Otherwise, the Content Manager for iSeries system ignores this parameter. For more information on the ANNOTATIONSTRUCT structure, see “ANNOTATIONSTRUCT (Annotation Information Structure)” on page 134.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input
	Not supported.
<i>pRC</i>	PRCSTRUCT — input /output
	The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0.
<i>ulParam1</i>	Contains <i>hObj</i> , an HOBJ pointer to an object handle block.
<i>ulParam2</i>	If you specified SIM_OPEN as a flag in the <i>fCreateControl</i> parameter and the field is not NULL, it contains the object access handle. This handle has the data type HOBJACC. The value in this field identifies the current instance of the accessed object.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_INVALID_OPTIONS • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_LOCAL_STORAGE_MODE • SIM_RC_INVALID_OBJECT_HANDLE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_SMS_PTR • SIM_RC_NOT_SUPPORTED • SIM_RC_OBJECT_ALREADY_EXISTS • SIM_RC_OPEN_FAILED • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Preparation:

- The object that you catalog must exist as a file.
- To get the defined values for the *ulConCls* parameter, use the **Ip2ListContentClasses** function.

Effects:

- This function creates an object and writes to that object the contents of the file that you specify.
- On successful completion, this function returns an object handle that you can use to access the object.

Your input values in the HOBJ data structure affect the results of this function. Input values for the *szItemID*, *ulPart*, and *chRepType* fields in that structure are optional.

If 0 is specified for the part number, the next sequential part number is created. If part number is nonzero, that part number is used if it does not already exist. If it does exist, the first available number is returned. Part number 1 is typically a base part. This API lets you create part number 2 – for example, a note – before creating part number 1.

- If you do not specify the SIM_OPEN flag for the *fCreateControl* parameter, the object is closed, but you can open it using the **SimLibOpenObject** function. Then you can access the object by using the object access handle that the function returns. You must use the object handle when referencing this object.
- Although your application can store its own affiliated types, other applications may not be able to process those objects.

SimLibCatalogObject

Exceptions: The content class parameter is not validated as a defined, known content class.

Follow-Up Tasks:

- If you specify SIM_OPEN, close the object when you finish with it, using the **SimLibCloseObject** function.
- After you finish using the pointer to the object handle block, free its space by using the **SimLibFree** function.

Example

```
#include <stdio.h> /* Standard I/O header files */
#include <string.h> /* Standard string header file */
#include "ekdviapi.h" /* Content Manager for iSeries */

main ()
{
    HSESSION hSession; /* from logon
    HOBJ hObj;
    HOBJ hObj2; /*get pointer from catalog
    ULONG ulConCls = SIM_CC_MODCA_IS2; /* mod:ca object
    SMS sms;
    CHAR pszFullFileName[45];
    UCHAR ulPriority = 0; /* not supported
    BITS fCreateControl = SIM_OPEN; /*leave open-get hobjacc
    ULONG ulVersion = 0; /* not supported
    LONG lSeqAfterPart = 0; /* take default
    ULONG ulAffiliatedType = SIM_BASE; /* base part
    PVOID pAffiliatedData = NULL; /* no affil data for base part
    RCSTRUCT RC;
    PRCSTRUCT pRC = &RC;
    POBJ pObj; /* Created object handle
    HOBJACC hObjAcc; /* object access handle
    USHORT sResult; /* return codes
    // create hobj
    if(0==( pObj=malloc(sizeof(OBJ)))) {
        return(1);
    }
    ( pObj)->ulStruct = sizeof(OBJ);
    strcpy(( pObj)->szItemID,"");
    strcpy(( pObj)->chRepType,"");
    ( pObj)->ulPart = 0;
    hObj = pObj;

    strcpy(pszFullFileName, "d:\\spid\\modca.mda");
    memset(SMS,0, sizeof(sms)); /* null out struct to get defaults
    strcpy(SMS.szCollectionName, "DFT");

    sResult = SimLibCatalogObject(
        hSession,
        hObj,
        ulConCls,
        SMS,
        pszFullFileName,
        ulPriority,
        fCreateControl,
        ulVersion,
        lSeqAfterPart,
        ulAffiliatedType,
        pAffiliatedData,
        0,
        pRC);
    if (pRC ->ulRC == SUCCESS) {
        // When only HOBJ is returned, it is in ulParam1
        hObj2 = (HOBJ)pRC->ulParam1;
    }
}
```

```

// Free memory allocated for HOBJ
SimLibFree(hSession, (PVOID)(hObj2), pRC);
// Mem containing the HOBJACC struct is freed by SimLibCloseObject.
hObjAcc = pRC->ulParam2; // object access handle
}
}

```

Related Functions

- Ip2ListContentClasses
- SimLibCloseObject
- SimLibCreateItem
- SimLibCreateObject
- SimLibFree
- SimLibOpenObject
- SimLibWriteObject

SimLibChangeIndexClass (Change the Index Class for an Item)

Format

```
SimLibChangeIndexClass( hSession, hItem, usClassId, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibChangeIndexClass** function to change the index class of an item to the index class that you specify.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hItem</i>	HITEM — input The handle to a virtual item. The SimLibOpenItemAttr function returns this handle.
<i>usClassId</i>	USHORT — input The identifier of the index class to change to.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.

SimLibChangeIndexClass

- ulRC* Contains one of the following return codes:
- SIM_RC_OK
 - SIM_RC_COMPLETION_ERROR
 - SIM_RC_INVALID_HITEM_VALUE
 - SIM_RC_INVALID_HSESSION
 - SIM_RC_INVALID_PASSED_ATTRIBUTE_DATA
 - SIM_RC_INVALID_PATTRIBUTE_PTR
 - SIM_RC_INVALID_POINTER
 - SIM_RC_INVALID_PRC
 - SIM_RC_INVALID_USATTRIBUTEID_VALUE
 - SIM_RC_INVALID_USCLASSID_VALUE
 - SIM_RC_NO_WRITE_ACCESS
 - SIM_RC_OUT_OF_MEMORY
 - SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Preparation: Before you can use this function, you must use **SimLibOpenItemAttr** to open the item for write access.

Effects:

- By changing the index class of an item, this function associates a different user-defined attribute set with that item.
- If the item is not open for write access, the function returns error SIM_RC_NO_WRITE_ACCESS.
- If the function fails, the Content Manager for iSeries system maintains the current attribute set for this item.
- If any index class attributes are common to both the original index class and the new one you specify for the item, the function copies those attributes to the new index class. Your application can then use the **SimLibWriteAttr** function to set the new index class attributes to the values you want. After you specify all the required attribute values for the new index class, you can make these values permanent by saving changes to the item using **SimLibSaveAttr** or **SimLibCloseAttr**.
- Use **SimLibGetClassInfo** to determine the attributes associated with an index class and **SimLibGetAttrInfo** to get details about an attribute.
- **SimLibOpenItemAttr** does not validate if the user has SIM_ACCESS_READ_WRITE authority. This authority is validated when **SimLibCloseAttr** is called with the SIM_OPT_SAVE parameter.

Related Functions

- **SimLibCloseAttr**
- **SimLibGetAttrInfo**
- **SimLibGetClassInfo**
- **SimLibOpenItemAttr**
- **SimLibSaveAttr**
- **SimLibWriteAttr**

SimLibChangeObjectSMS (Change the SMS Criteria for an Object)

Format

```
SimLibChangeObjectSMS( hSession, hObj, pSMS, fChangeControl, pAsyncCtl,
pRC )
```

Purpose

Use the **SimLibChangeObjectSMS** function to modify the system-managed storage (SMS) criteria for an object.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObj</i>	HOBJ — input The pointer to an object handle block in the HOBJ data structure. For more information on the HOBJ structure, see “HOBJ (Handle to Query Stored Object)” on page 143.
<i>pSMS</i>	PSMS — input Pointer to a system-managed storage (SMS) structure for an object. This structure uses only <i>szCollectionName</i> .
<i>fChangeControl</i>	BITS — input Not supported.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_FOPTIONS • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_ITEMID • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC

- SIM_RC_INVALID_PSMS_VALUE
- SIM_RC_INVALID_SMS_PTR
- SIM_RC_NEW_COLLECTION_NOT_FOUND
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PART_NOT_FOUND
- SIM_RC_PRIVILEGE_ERROR

Related Functions

- SimLibCreateObject
- SimLibQueryObject

SimLibCloseAttr (Close an Attribute Set)

Format

```
SimLibCloseAttr( hSession, hItem, ulDisposition, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibCloseAttr** function to release the access rights that your application has to the folder or document you specify. You can use this function to replace the permanent attributes of the item in the database with modifications that have been made to the virtual item. Alternatively, you can use this function to discard modifications to the virtual item without updating the permanent attributes.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hItem</i>	HITEM — input The handle to a virtual item. The SimLibOpenItemAttr function returns this handle.
<i>ulDisposition</i>	ULONG — input The action to take regarding modifications to the item. The value of this parameter determines whether the Content Manager for iSeries system saves or discards modifications to the attributes of the virtual item. If the item is accessed for reading only or if none of its attributes are changed, the Content Manager for iSeries system ignores this parameter. The valid values are: SIM_OPT_SAVE Updates the permanent attributes of the item in the database by using the current attribute settings of the virtual item. All required attributes of the index class must be written before closing, or the function returns the error SIM_RC_REQUIRED_ATTRIBUTE_MISSING. This value is valid only if the item is open for update. SIM_OPT_DISCARD Discards modifications to the attribute settings of the virtual item without updating the permanent attributes of the item in the database.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_FOPTIONS • SIM_RC_INVALID_HITEM_VALUE • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_USACCESSLEVEL_VALUE • SIM_RC_INVALID_USCLASSID_VALUE • SIM_RC_INVALID_USDISPOSITION_VALUE • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR • SIM_RC_REQUIRED_ATTRIBUTE_MISSING

Guidelines for Use

Effects: The function closes the virtual attribute set and you can no longer use the access handle. The function also frees the space used by the access handle.

Related Functions

-
- SimLibChangeIndexClass
- SimLibOpenItemAttr
- SimLibSaveAttr
- SimLibWriteAttr

SimLibCloseObject (Close an Object)

Format
SimLibCloseObject(*hSession, hObjAcc, fCommit, pAsyncCtl, pRC*)

Purpose

Use the **SimLibCloseObject** function to close an open object and end access to that object.

You must use this function to close objects that you opened using any of the following functions:

- **SimLibCatalogObject**

SimLibCloseObject

- **SimLibCreateObject**
- **SimLibOpenObject**

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObjAcc</i>	HOBJACC — input The object access handle. The value of this parameter identifies the current instance of the accessed object.
<i>fCommit</i>	BOOL — input Not supported.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_FOPTIONS• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_OBJECT_ACCESS_HANDLE• SIM_RC_INVALID_OBJECT_HANDLE• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_OUT_OF_MEMORY• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects: After successful completion of the function, you can no longer use the access handle. The function also frees the space used by the access handle, so **SimLibFree** should not be called.

If SIM_RC_PRIVILEGE_ERROR is returned, you must call **SimLibCloseAttr** using SIM_OPT_DISCARD to guarantee that the item lock has been released.

Example

```

#include <stdio.h>                /* Standard I/O header files    */
#include "ekdviapi.h"            /* Content Manager for iSeries  */

main ()
{
    HSESSION hSession;           // get from logon
    HOBJACC hObjAcc;             // get from catalog, open, or create
    BOOL fCommit = TRUE;        // keep the changes
    RCSTRUCT RC;
    PRCSTRUCT pRC = &RC;
    USHORT sResult;              // return codes

    /*Call the function */

    sResult = SimLibCloseObject(
        hSession,
        hObjAcc,
        fCommit,
        0,
        pRC);
}

```

Related Functions

- SimLibCatalogObject
- SimLibCreateObject
- SimLibOpenObject

SimLibCopyObject (Copy an Object)

Format

```

SimLibCopyObject( hSession, hDestObj, hSrcObj, pSMS, ulPriority, fDelete,
pAsyncCtl, pRC )

```

Purpose

Use the **SimLibCopyObject** function to copy an entire object from a source object location to a target object location, replacing an existing target object. Neither the source object nor the target object can currently be open.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hDestObj</i>	HOBJ— input The destination object handle. The value of this parameter identifies the target object.
<i>hSrcObj</i>	HOBJ— input The source object handle. The value of this parameter identifies the source object that the function copies.
<i>pSMS</i>	PSMS— input Not supported.
<i>ulPriority</i>	ULONG— input

SimLibCopyObject

	Not supported.
<i>fDelete</i>	BOOL— input
	Not supported.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT— input
	Not supported.
<i>pRC</i>	PRCSTRUCT — input/output
	The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0. If the return code is SIM_RC_ITEM_CHECKEDOUT, this field contains the value 1 to indicate that ulParam1 contains a pointer. If the Content Manager for iSeries system returns any other error, this field contains the value NULL.
<i>ulParam1</i>	Contains the value NULL if the return code is SIM_RC_ITEM_CHECKEDOUT.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INUSE• SIM_RC_INVALID_FOPTIONS• SIM_RC_INVALID_HSESSION• SIM_RC_NOT_SUPPORTED_• SIM_RC_OUT_OF_MEMORY• SIM_RC_PRIVILEGE_ERROR

Related Functions

- SimLibLogon

SimLibCreateItem (Create an Item)

Format

```
SimLibCreateItem( hSession, usItemType, usIndexClass, usNumOfAttrs,  
pAttributeList, ulAccessControl, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibCreateItem** function to create a new document or a new folder in the index class that you specify. You must specify any required attributes for that index class. You can also specify optional attributes for the item.

Parameters

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

<i>usItemType</i>	USHORT — input The type of item you want to create. The valid values are: SIM_DOCUMENT Indicates that the item is a document. SIM_FOLDER Indicates that the item is a folder.
<i>usIndexClass</i>	USHORT — input An index class identifier for the set of user-defined attributes to associate with this item. This index class must exist at the time you log on. If you do not require any user-defined attributes, use SIM_INDEX_NOINDEX , which is a special index class created during installation and preset with user-defined attributes, to indicate that the item has not yet been indexed. “Guidelines for Use” explains why it is important to use a predefined index class.
<i>usNumOfAttrs</i>	USHORT — input The number of data structures in the <i>pAttributeList</i> parameter array.
<i>pAttributeList</i>	PATRLISTSTRUCT — input The pointer to an array of ATTRLISTSTRUCT data structures that contain the attributes to associate with this document or this folder. Each data structure in the array specifies one attribute. If you set this parameter to NULL, no attributes are associated with the item. For more information on the ATTRLISTSTRUCT data structure, see “ATTRLISTSTRUCT (Attribute List Data Structure)” on page 137. To add attributes to the item later, your application must first open the item and then use separate functions to write the attributes to it.
<i>ulAccessControl</i>	ULONG — input Not supported.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer. If an error occurs, this field contains the value 0.
<i>ulParam1</i>	Contains a PITEMID pointer to a buffer with the item identifier (<i>pszItemID</i>) for the new item.
<i>ulParam2</i>	The function does not use this field.

SimLibCreateItem

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_ATTR_NOT_FOUND
- SIM_RC_ATTRIBUTE_READ_ONLY
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_INDEX_CLASS
- SIM_RC_INVALID_MSGID
- SIM_RC_INVALID_PASSED_ATTRIBUTE_DATA
- SIM_RC_INVALID_PATTRIBUTELIST_PTR
- SIM_RC_INVALID_PATTRIBUTELIST_VALUE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_USATTRIBUTEID_VALUE
- SIM_RC_INVALID_USITEMTYPE_VALUE
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Preparation:

- Use of a predefined index class is important so that you can use the **SimLibSearch** function to locate items.
- To add an item to a newly created index class, log off and then log on again before using this function, so that the index class is in existence at logon time.
- You can also create items automatically by using the **SimLibCatalogObject** or **SimLibCreateObject**. Use **SimLibCreateItem** when you have an index class with attribute values. Then use **SimLibCatalogObject**, **SimLibCreateObject**, or **SimLibStoreNewObject** to put objects into the new item.

Follow-Up Tasks: After the function gets the item identifier, use the **SimLibFree(hSession, (PVOID)ulParam1, pRC)** function to free the buffer.

Example

```
#include <windows.h>                /* Main Windows header files */
#include <sys\types.h>
#include <stdio.h>                  /* Standard I/O header files */
#include <stdlib.h>                 /* Standard library header files*/
#include <stdarg.h>
#include <stddef.h>
#include <io.h>
#include "ekdviapi.h"              /* Content Manager for iSeries */

main ()
{
    HSESSION    hSession;           /* Product session handle */
    ITEMID      FolderItemID;      /* ItemID of new folder */
    USHORT      usFoldAttrs;       /* Number of ATTRLISTSTRUCTs */
    ATTRLISTSTRUCT Folder [ 1 ] = {
        sizeof(Folder),           /* structure size */
        "SourceName",             /* attribute value */
        SIM_ATTR_READWRITE,       /* attribute flags */
        140,                      /* attribute ID */
        SIM_ATTR_FSTRING           /* attribute type */
    };

    USHORT      usIndexClass;       /* Index class for folder */
    RCSTRUCT     RCStruct;          /* RC data structure */
}
```



```

USHORT      sResult;          /* return codes          */

/*****
/* Initialize SimLibCreateItem Parameters.          */
*****/

/* We will create an item in the SIM_INDEX_NOINDEX Index Class. */
/* This index has three optional attributes. We will provide a */
/* value for only one of these attributes. This is done by     */
/* initializing the attribute array "Folder" above.            */

usIndexClass = SIM_INDEX_NOINDEX; /* Index Class of the folder */
usFoldAttrs  = 1;                 /* # of attrs for the folder */

/*****
/* Call SimLibCreateItem to create a new folder          */
*****/

sResult = SimLibCreateItem(
    hSession,          /* session handle from SimLibLogon*/
    SIM_FOLDER,       /* Create a folder                */
    usIndexClass,     /* Index class of folder          */
    usFoldAttrs,     /* Number of attribute lists      */
    &Folder,          /* Pointer to attribute list      */
    NULL,             /* Reserved for future use       */
    NULL,             /* Request SYNCHRONOUS processing*/
    &RCStruct         /* Pointer to RC data structure*/
);

/*****
/* If successful, copy the itemID                      */
*****/

if (sResult == SIM_RC_OK) {
    strcpy (FolderItemID, (char*)RCStruct.ulParam1;
    printf("New Folder ItemID      = %s\n\n", FolderItemID);
}
else {
    /* ..... exception processing ..... */
}
}

```

Related Functions

- SimLibChangeIndexClass
- SimLibFree
- SimLibGetAttrInfo
- SimLibGetClassInfo
- SimLibSearch

SimLibCreateObject (Create an Object)

Format

```

SimLibCreateObject( hSession, hObj, ulConCls, pSMS, ulPriority, fCreateControl,
ulVersion, lSeqAfterPart, ulAffiliatedType, pAffiliatedData, pAsyncCtl, pRC )

```

Purpose

Use the **SimLibCreateObject** function to create a new empty object, such as when your data is in memory rather than in a file.

SimLibCreateObject

You can also create an object using the **SimLibCatalogObject** function, which is equivalent to using the **SimLibCreateObject**, **SimLibWriteObject**, and **SimLibCloseObject** functions. You can also create an object using the **SimLibStoreNewObject** function, which is simpler than using the combination of functions.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObj</i>	HOBJ — input The pointer to an object handle block in the HOBJ data structure. For more information on the HOBJ data structure, see “HOBJ (Handle to Query Stored Object)” on page 143. “Guidelines for Use” describes the effects of your input to this data structure.
<i>ulConCls</i>	ULONG — input The content class identifier for the object. The value of this parameter tells what kind of data is in the object that you are creating (see Appendix B, “Predefined Content Classes,” on page 295). To indicate an undefined content class, specify the value SIM_CC_UNKNOWN for this parameter. However, if you do not use a defined content class, other applications cannot use Content Manager for iSeries content class services to determine how to manipulate the contents of the objects that you store.
<i>pSMS</i>	PSMS — input Pointer to a system-managed storage (SMS) structure for an object. This structure uses only <i>szCollectionName</i> .
<i>ulPriority</i>	ULONG — input Not supported.
<i>fCreateControl</i>	BITS — input Control option bits for the creation operation. The valid values are: SIM_CLOSE Closes the object on completion of the request. This is the default. SIM_OPEN Leaves the object open in update mode. If you do not specify this flag, the created object is closed.
<i>ulVersion</i>	ULONG — input Not supported.
<i>lSeqAfterPart</i>	LONG — input Not supported.
<i>ulAffiliatedType</i>	ULONG — input The type of affiliated object. The defined values are:

SIM_ANNOTATION

Indicates that the object is an annotation associated with a folder or a document.

SIM_BASE

Indicates that the object is a base object such as a MO:DCA or TIFF file, and is not an annotation, note, or event associated with a folder or document.

SIM_EVENT

Indicates that the object is an event associated with a folder or a document.

SIM_MGDS

Indicates that the object is an MGDS (machine-generated data stream) associated with a folder or a document.

SIM_NOTE

Indicates that the object is a note associated with a folder or a document.

pAffiliatedData PVOID — input

The pointer to a data structure of the type ANNOTATIONSTRUCT. If the *ulAffiliatedType* parameter contains the value SIM_ANNOTATION, *pAffiliatedData* points to this structure, which contains additional data affiliated with the object. Otherwise, the Content Manager for iSeries system ignores this parameter. For more information on the ANNOTATIONSTRUCT structure, see “ANNOTATIONSTRUCT (Annotation Information Structure)” on page 134.

pAsyncCtl PASYNCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 0.

ulParam1 Contains *hObj*, an HOBJ pointer to an object handle block.

ulParam2 If the *fCreateControl* parameter flag was set to SIM_OPEN and this field is not null, it contains *hobjacc*, the object access handle. This handle has the data type HOBJACC. The value in this field identifies the current instance of the accessed object.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE
- SIM_RC_INVALID_OBJECT_HANDLE
- SIM_RC_INVALID_POINTER

SimLibCreateObject

- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_SMS_PTR
- SIM_RC_OPEN_FAILED
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR
- SIM_RC_INVALID_USCLASSID_VALUE

Guidelines for Use

Preparation: To get the supported values for the *ulConCls* parameter, use the **Ip2ListContentClasses** function.

Effects:

- This function creates an empty object that you can write to using **SimLibWriteObject**.
- On successful completion, this function returns an object handle that you can use to access the object.
- You can create a new object within a specified item or create both the item and an object within it. If you create the item, you cannot specify any attributes. The item is placed in the SIM_INDEX_NOINDEX index class. You must do that later using the **SimLibOpenItemAttr**, **SimLibWriteAttr**, and **SimLibCloseAttr** functions.
- Although your application can store its own affiliated types, other applications may not be able to process those objects.
- Your input values in the HOBJ data structure affect the results of this function. Input values for the *szItemID*, *ulPart*, and *chRepType* fields in this structure are optional.
If 0 is specified for the part number, the next sequential part number is created. If part number is nonzero, that part number is used if it does not already exist. If it does exist, the first available number is returned. Part number 1 is typically a base part. This API lets you create part number 2 – for example, a note – before creating part number 1.
- If the function closed the object, you can open it using the **SimLibOpenObject** function.
- If the function returns the object access handle, this handle identifies the current instance of access to the open object. This handle is different from the handle normally used to reference the stored object. Use the object access handle (*hObjAcc*), not the object handle (*hObj*), with the following functions:
 - **SimLibCloseObject**
 - **SimLibReadObject**
 - **SimLibResizeObject**
 - **SimLibSeekObject**
 - **SimLibWriteObject**

Exceptions:

- The content class parameter is not validated as a defined, known content class.

Follow-Up Tasks:

- After your application finishes with *hObj*, the object handle, free the space by using the **SimLibFree** function.
- Your application should not free the space used by *hObjAcc*, the object access handle, because the later call to **SimLibCloseObject** frees the space.

Example

```

#include <stdio.h>                /* Standard I/O header files      */
#include <string.h>              /* Standard string header file    */
#include "ekdviapi.h"           /* Content Manager for iSeries    */

main()
{
    HSESSION hSession;           // get from logon
    HOBJ hObj, hObj2;
    ULONG ulConCls = SIM_CC_MODCA_IS2; // mod:ca object
    SMS sms;
    ULONG ulPriority = 0;        // not supported
    BITS fCreateControl = SIM_OPEN; //leave open-get hobjacc
    ULONG ulVersion = 0;        // not supported
    LONG lSeqAfterPart = 0;     // not supported
    ULONG ulAffiliatedType = SIM_BASE;
    PVOID pAffiliatedData = NULL; // no affiliated data
    RCSTRUCT RC;
    PRCSTRUCT pRC = &RC;
    POBJ pObj;                  // Created object handle
    USHORT sResult;             // get rc back
    HOBJACC hObjAcc;            // object access handle

    // create hobj
    if(0==( pObj=(POBJ) malloc(sizeof(OBJ)))) {
        return(1);
    }
    ( pObj)->ulStruct = sizeof(OBJ);
    strcpy(( pObj)->szItemID,"");
    strcpy(( pObj)->chRepType,"");
    ( pObj)->ulPart = 0;
    hObj = pObj;

    memset(SMS,0, sizeof(sms)); // null out struct to get defaults
    strcpy(SMS.szCollectionName, "*DFT");

    /*Call the function*/

    sResult = SimLibCreateObject(
        hSession,
        hObj,
        ulConCls,
        SMS,;
        ulPriority,
        fCreateControl,
        ulVersion,
        lSeqAfterPart,
        ulAffiliatedType,
        pAffiliatedData,
        0,
        pRC);
    if (pRC ->ulRC == SUCCESS) {
        // When only HOBJ is returned, it is in ulParam1
        hObj2 = (HOBJ)pRC->ulParam1;
        // Free memory allocated for HOBJ
        SimLibFree(hSession, (PVOID)(hObj2), pRC);
        // Mem containing the HOBJACC struct is freed by SimLibCloseObject.
        hObjAcc = pRC->ulParam2; // object access handle
    }
}

```

Related Functions

- [Ip2ListContentClasses](#)
- [SimLibCatalogObject](#)

SimLibCreateObject

- SimLibCloseObject
- SimLibCreateObject
- SimLibFree
- SimLibOpenObject
- SimLibReadObject
- SimLibResizeObject
- SimLibSeekObject
- SimLibWriteObject

SimLibDeleteItem (Delete an Item)

Format

```
SimLibDeleteItem( hSession, pszItemID, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibDeleteItem** function to delete a folder or a document from the system.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemID</i>	PITEMID — input The identifier of an item you want to delete. This identifier is the item ID.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0. If the item is locked on the server, this field contains the value 1, to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	If <i>usParam</i> is 1, this field contains a pointer to a buffer with a USERACCESSSTRUCT data structure. This data structure contains a user ID that indicates who has locked the item. If any other error is returned, this field contains the value NULL.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INUSE• SIM_RC_INVALID_HSESSION

- SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE
- SIM_RC_INVALID_PITEMIDITEM_PTR
- SIM_RC_INVALID_PITEMIDITEM_VALUE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_ITEM_CHECKEDOUT
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PARENT_CHECKEDOUT
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- This function removes the specified document or folder from the database. After completion of the function, the item ID (*pszItemID*) associated with the item is no longer valid.
- The function automatically removes any references to the deleted item in the table of contents of folders or workbaskets that list it.
- For either a folder or a document, the Content Manager for iSeries system deletes all objects associated with the item.
- If a folder is deleted, documents or folders in the folder are not deleted.

Exceptions:

- This function cannot delete an item if the item, or a folder containing the item, is currently locked by a user ID other than the one you specified on the *pszUserID* parameter when you used **SimLibLogon** to begin this Content Manager for iSeries session.

A folder can have more than one parent folder. If a parent folder is locked and **SimLibDeleteItem** returns SIM_RC_PARENT_CHECKEDOUT, the function does not identify the folder that is locked.

Follow-Up Tasks: After your application no longer needs the user access information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer containing the USERACCESSSTRUCT data structure.

Example

```
#include <windows.h>                /* Main Windows header files */
#include <sys\types.h>
#include <stdio.h>                  /* Standard I/O header files */
#include <stdlib.h>                 /* Standard library header files*/
#include <stdarg.h>
#include <stddef.h>
#include <io.h>
#include "ekdviapi.h"              /* Content Manager for iSeries */
main ()
{

    HSESSION    hSession;          /* Product session handle */
    PITEMID     pszItemID;         /* Pointer to an item ID. */
    RCSTRUCT    RCStruct;         /* RC data structure */
    USHORT      sResult;          /* return codes */

    /******
    /*Initialize the itemID to prepare for a call to SimLibDeleteItem*/
    /******
    memset (pszItemID, '\0', DOC_ID_SIZE); /* set to null */
    strcpy ((CHAR *)pszItemID, (CHAR *) "DA97220AA.AAB");

    /******
```

SimLibDeleteItem

```
/* Call SimLibDeleteItem to delete a document from the system */
/*****/

sResult = SimLibDeleteItem(
    hSession,          /* session handle from SimLibLogon */
    pszItemID,        /* itemID to be deleted */
    (PASYNCTLSTRUCT) NULL, /* Request SYNCHRONOUS processing*/
    (PRCSTRUCT) &RCstruct /* Pointer to RC data structure */
);

if (sResult != SIM_RC_OK) {
    printf("Item %s cannot be deleted", pszItemID);
}
}
```

Related Functions

- **SimLibAddFolderItem**
- **SimLibCloseAttr**
- **SimLibCreateItem**
- **SimLibFree**
- **SimLibGetItem**
- **SimLibOpenItemAttr**

SimLibDeleteObject (Delete an Object)

Format

```
SimLibDeleteObject( hSession, hObj, ulDeleteOption, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibDeleteObject** function to delete the object that you specify.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObj</i>	HOBJ — input The pointer to an object handle block in the HOBJ data structure. For more information on the HOBJ structure, see “HOBJ (Handle to Query Stored Object)” on page 143.
<i>ulDeleteOption</i>	ULONG — input Not supported.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0. If the return code is SIM_RC_ITEM_CHECKEDOUT, this field contains the value 1, to indicate that <i>ulParam1</i> contains a pointer. If the Content Manager for iSeries system returns any other error, this field contains the value NULL.
<i>ulParam1</i>	If <i>usParam</i> is 1, this field contains a pointer to a USERACCESSSTRUCT data structure. The data structure contains the user ID of the user who has locked the item.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_OBJECT_HANDLE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_ITEM_CHECKEDOUT • SIM_RC_ITEM_NOT_FOUND • SIM_RC_OUT_OF_MEMORY • SIM_RC_PART_NOT_FOUND • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects: When the last object in an item is deleted, the item is also deleted. To delete all the objects in one operation, use **SimLibDeleteItem**, which deletes the item and all the objects within it.

Exceptions:

- You cannot delete an object if the item that contains the object is locked by someone else.
- If the item contains only the object, the item is also deleted.

SimLibFree (Free Memory)

Format

```
SimLibFree( hSession, pBuffer, pRC )
```

Purpose

Use the **SimLibFree** function to free all memory allocated and returned by the Content Manager for iSeries system. Do not call this function if your application allocated the memory. Use it only as directed.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pBuffer</i>	PVOID — input A pointer to a data structure of indeterminate type.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in the RCSTRUCT data structure:

usParam The function does not use this field.

ulParam1 The function does not use this field.

ulParam2 The function does not use this field.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC

Example

```

ULONG          ulRC;
HSESSION      hsession;
RCSTRUCT      RC;

ulRC = SimLibListClasses(hSession, 0, NULL, &RC);
if (ulRC == SIM_RC_OK) {
    // process list of classes
    SimLibFree(hSession, (PVOID)RC.ulParam1, &RC);
}
    
```

Related Functions

- SimLibLogon

SimLibGetAttrInfo (Get Attribute Information)

Format
SimLibGetAttrInfo(*hSession*, *usAttributeId*, *pAsyncCtl*, *pRC*)

Purpose

Use the **SimLibGetAttrInfo** function to return detailed information for a specific attribute in the system. This function can return information for both the system-defined attributes and the user-defined index attributes.

Parameters

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

usAttributeId USHORT — input

The unique identifier assigned to an attribute. You can pass the ID of an index class or one of the following Content Manager for iSeries system-defined attributes:

OIM_ID_ITEM_CREATE_TIMESTAMP

Indicates the creation time of the item.

OIM_ID_ITEM_NAME

Indicates the name of the item. This attribute is optional.

OIM_ID_SYS_MOD_TIMESTAMP

Indicates the last time the item was changed.

OIM_ID_UID

Indicates the item ID.

pAsyncCtl PASYNCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 1, to indicate that *ulParam1* contains a pointer. If completion is not successful, this field contains the value 0.

ulParam1 Contains a pointer to a buffer where an ATTRINFOSTRUCT data structure provides information about the specified attribute. For more information on the ATTRINFOSTRUCT data structure, see “ATTRINFOSTRUCT (Attribute Information Structure)” on page 135.

ulParam2 The function does not use this field.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_USATTRIBUTEID_VALUE
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the ATTRINFOSTRUCT data, use the **SimLibFree**(*hSession*,(PVOID)*ulParam1*, *pRC*) function to free the buffer containing the structure.

Related Functions

- Ip2ListAttrs
- SimLibFree
- SimLibGetClassInfo

SimLibGetClassInfo (Get Index Class Information)

Format

```
SimLibGetClassInfo( hSession, usClassType, usID, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetClassInfo** function to return detailed information for a specific index class defined in the system.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>usClassType</i>	USHORT— input The type of information that the <i>usID</i> parameter contains. The valid values are: SIM_INDEXCLASSID Indicates that the <i>usID</i> parameter contains an index class ID.
<i>usID</i>	USHORT — input The ID of an index class.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer to the data area.
<i>ulParam1</i>	Contains a pointer to a buffer with a CLASSINFOSTRUCT data structure.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_CLASS_TYPE • SIM_RC_INVALID_FOPTIONS • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_USCLASSID_VALUE

- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Exceptions: The information that this function returns is subject to access control restrictions. If you do not have access to the index class, the function fails and SIM_RC_INVALID_USCLASSID_VALUE is returned.

Follow-Up Tasks: When your application no longer needs the CLASSINFOSTRUCT data, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

SimLibGetItemAffiliatedTOC (Get a Table of Contents for Item Affiliates)

Format

```
SimLibGetItemAffiliatedTOC( hSession, pszItemID, usAffiliatedType, pAsyncCtl,  
pRC )
```

Purpose

Use the **SimLibGetItemAffiliatedTOC** function to get a table of contents that lists the affiliated objects for an item.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemID</i>	PITEMID — input The identifier of an item for which you want a table of contents listing affiliated objects. This identifier is the item ID.
<i>usAffiliatedType</i>	USHORT — input The type of affiliated object to list in the table of contents. The valid values are: SIM_ANNOTATION Lists annotations associated with the folder or document. SIM_BASE Lists base objects, such as MO:DCA or TIFF files, that are not annotations, notes, or events associated with the folder or document. SIM_EVENT Lists events associated with the folder or document. SIM_MGDS Lists MGDS (machine-generated data streams) associated with the folder or document. SIM_NOTE Lists notes associated with the folder or document.

SIM_ALL

Lists all types of objects associated with the folder or document.

If you specify that you want to return objects other than base objects, they must have a nonzero length. Base objects are always included regardless of their length.

pAsyncCtl PASYNCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 1, to indicate that *ulParam1* contains a pointer. Otherwise, this field contains the value 0.

ulParam1 Contains a pointer to a buffer with an array of AFFTOCENTRYSTRUCT data structures. If no affiliated objects satisfy the *usAffiliatedType* filter, this field contains the value NULL. For more information on the AFFTOCENTRYSTRUCT data structure, see “AFFTOCENTRYSTRUCT (Affiliated Table of Contents Entry Structure)” on page 133.

ulParam2 Contains the number of entries in the AFFTOCENTRYSTRUCT array referenced by *ulParam1*. If no affiliated objects satisfy the *usAffiliatedType* filter, this field contains the value NULL.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_COMPLETION_MSG_NOT_POSTED
- SIM_RC_COMPLETION_SEM_ALREADY_POSTED
- SIM_RC_COMPLETION_SEM_TOO_MANY_POSTS
- SIM_RC_DOCSS_ERROR
- SIM_RC_ERROR_RELEASING_SEMAPHORE
- SIM_RC_ERROR_REQUESTING_SEMAPHORE
- SIM_RC_FUNC_NOT_IN_TRANS
- SIM_RC_GETRESPONSE_TIMEOUT
- SIM_RC_INVALID_AFFILIATEDTYPE_VALUE
- SIM_RC_INVALID_PITEMIDITEM_PTR
- SIM_RC_INVALID_PITEMIDITEM_VALUE
- SIM_RC_INVALID_PLATSESSION_TYPE
- SIM_RC_INVALID_PRC
- SIM_RC_ITEM_NOT_FOUND
- SIM_RC_NOT_SUPPORTED
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Follow-Up Tasks: After you get the TOC information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to clear the buffer containing the AFFTOCENTRYSTRUCT data structures.

Related Functions

- **SimLibFree**
- **SimLibLogon**

SimLibGetItemInfo (Get Item Information)

Format

```
SimLibGetItemInfo( hSession, pszItemID, usClassId, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetItemInfo** function to return the following information about a document or a folder to your application:

- Item type
- Item name
- Index class of the item
- Workflow information
- User ID of anyone who has locked the item

Parameters

<i>hSession</i>	HSESSION — input	The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemID</i>	PITEMID — input	The identifier of an item for which you want information. This identifier is the item ID.
<i>usClassId</i>	USHORT — input	The identifier of an index class.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input	Not supported.
<i>pRC</i>	PRCSTRUCT — input/output	The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer to a data area.
<i>ulParam1</i>	Contains a pointer to an ITEMINFOSTRUCT data structure that provides the item information. For more information on this data structure, see “ITEMINFOSTRUCT (Item Information Structure)” on page 144.

SimLibGetItemInfo

<i>ulParam2</i>	Contains the value 1, indicating that the buffer referenced by <i>ulParam1</i> contains 1 entry.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_ITEM_ID• SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE• SIM_RC_INVALID_ITEM_TYPE• SIM_RC_INVALID_PITEMIDITEM_PTR• SIM_RC_INVALID_PITEMIDITEM_VALUE• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_OUT_OF_MEMORY• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Exceptions: Do not use this function to return information about a workbasket. To return workbasket information, use **SimWmGetWorkBasketInfo**.

Follow-Up Tasks: When your application no longer needs the item information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

Related Functions

- **SimWmGetWorkBasketInfo**
- **SimLibListClasses**

SimLibGetItemSnapshot (Get a Snapshot of Item Attributes)

Format

```
SimLibGetItemSnapshot( hSession, pszItemID, fReadAttrInd, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetItemSnapshot** function to return a copy of the attributes associated with a document or a folder. Your application can substitute this function for the following sequence of Content Manager for iSeries functions:

- **SimLibGetItemType**
- **SimLibOpenItemAttr**
- **SimLibReadAttr**
- **SimLibCloseAttr**

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemID</i>	PITEMID — input The identifier of an item. This identifier is the item ID.
<i>fReadAttrInd</i>	BITS — input

The type of attribute values to return. Here are the valid values. You can use a bitwise inclusive OR operator (|) to combine them.

SIM_SYSTEM_ATTR

Returns the system-defined attribute values for the document or the folder.

SIM_USER_ATTR

Returns the user-defined attribute values for the document or the folder.

SIM_WORK_ATTR

Returns the work management information for the document or the folder.

The function returns attribute values for the current view. The Content Manager for iSeries system gets system-defined and user-defined attribute values from the SNAPSHOTSTRUCT data structure and returns them in the *pAttr* field of the ICVIEWSTRUCT data structure. It returns priority attributes and work management information in the *pWmSnapshot* field of the SNAPSHOTSTRUCT data structure. “Guidelines for Use” contains more detail. For more information on the ICVIEWSTRUCT and SNAPSHOTSTRUCT data structures, see “ICVIEWSTRUCT (Index Class View Information Structure)” on page 143 and “SNAPSHOTSTRUCT (Snapshot Information Structure)” on page 155.

pAsyncCtl PASYNCCTLSTRUCT — input
Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 1, to indicate that *ulParam1* contains a pointer to a data area.

ulParam1 Contains a pointer to a SNAPSHOTSTRUCT data structure that provides the returned attribute values.

ulParam2 The function does not use this field.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_ITEM_ID
- SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE
- SIM_RC_INVALID_ITEM_TYPE
- SIM_RC_INVALID_PITEMIDITEM_PTR
- SIM_RC_INVALID_PITEMIDITEM_VALUE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC

SimLibGetItemSnapshot

- SIM_RC_INVALID_READATTRIND
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR
- SIM_RC_SESSION_DB_VIEW_MISMATCH

Guidelines for Use

Exceptions: Your application might need to use a conversion routine such as an ASCII-to-integer routine to change the character representation of an attribute value into the correct form for the application.

Follow-Up Tasks: After your application has processed the information that the Content Manager for iSeries system returns to the SNAPSHOTSTRUCT data structure, use the **SimLibFree(hSession, (PVOID)ulParam1, pRC)** function to free the pointer to the SNAPSHOTSTRUCT data structure.

Related Functions

- SimLibCloseAttr
- SimLibFree
- SimLibGetItemType
- SimLibGetTOCData
- SimLibOpenItemAttr
- SimLibReadAttr

SimLibGetItemType (Get the Type of an Item)

Format

```
SimLibGetItemType( hSession, pszItemID, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetItemType** function to return the type of an item associated with the item identifier you specify.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemID</i>	PITEMID — input The identifier of an item for which you want to return the type. This identifier is the item ID.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the PRCSTRUCT structure, see “PRCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an PRCSTRUCT data structure:

<i>usParam</i>	Contains the value 0.
----------------	-----------------------

<i>ulParam1</i>	Contains one of the following values indicating the type of item: SIM_DOCUMENT Indicates that the item is a document. SIM_FOLDER Indicates that the item is a folder. SIM_WORKBASKET Indicates that the item is a workbasket. SIM_WORKFLOW Indicates that the item is a workflow.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_ID • SIM_RC_INVALID_PITEMIDITEM_PTR • SIM_RC_INVALID_PITEMIDITEM_VALUE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Effects: After successful completion of this function, you can use other Content Manager for iSeries functions to get additional detailed information about the item. To return additional information, use one of the following functions:

SimLibGetItemInfo

To return information about a folder or a document.

SimWmGetWorkBasketInfo

To return information about a workbasket.

Related Functions

- SimWmGetWorkBasketInfo
- SimLibGetItemInfo

SimLibGetItemXREF (Get a Cross-Reference for an Item)

Format

```
SimLibGetItemXREF( hSession, pszItemID, ulFilter, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetItemXREF** function to list the folders that contain the item you specify and match the other criteria you specify.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
-----------------	---

SimLibGetItemXREF

<i>pszItemID</i>	PITEMID — input The identifier of an item for which you want a cross reference. This identifier is the item ID.
<i>ulFilter</i>	ULONG — input The criteria to match for cross-referencing. Here are the valid values: SIM_XREF_FOLDERS_ONLY_FILTER Returns only folders that contain the specified item.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer. If no items match the criteria you specify, this field contains the value NULL.
<i>ulParam1</i>	Contains a pointer to a buffer with an array of ITEMID strings. Each string provides the item ID of a folder that contains the specified item. If no items match the criteria you specify, this field contains the value NULL.
<i>ulParam2</i>	Contains the number of entries pointed to by <i>ulParam1</i> .
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_ITEM_ID• SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE• SIM_RC_INVALID_ITEM_TYPE• SIM_RC_INVALID_PITEMIDITEM_PTR• SIM_RC_INVALID_PITEMIDITEM_VALUE• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_INVALID_USFILTER_VALUE• SIM_RC_OUT_OF_MEMORY• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Follow-Up Tasks: After you get the item ID information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer containing the cross-reference information.

SimLibGetSessionType (Get the Session Type)

Format

```
SimLibGetSessionType( hSession, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetSessionType** function to return information regarding the platform type of the current session.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in the RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a PSZ to the current session type. If you have a LAN-based library session, the session type is Ip2. Other values are platform dependent.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_INVALID_HSESSION • SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the session type information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

Related Functions

- SimLibLogon

SimLibGetTOC (Get a Table of Contents)

Format

```
SimLibGetTOC( hSession, pszItemID, usItemType, usWipFilter, usSuspendFilter,  
usNbrOfClasses, pusClassIdList, pLinkCriteria, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetTOC** function to return either a partial or a complete table of contents for the workbasket or folder you specify. The table of contents contains a list of the documents and folders in that workbasket or folder. You can specify a variety of values for the parameters of this function to determine the entries in the table of contents.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemID</i>	PITEMID — input The identifier of workbasket or folder for which you want a table of contents. This identifier is the item ID.
<i>usItemType</i>	USHORT — input The type of item to return in the table of contents. The valid values are: SIM_DOCUMENT Returns documents. SIM_FOLDER Returns folders. SIM_ALL Returns both documents and folders.
<i>usWipFilter</i>	USHORT — input Not supported.
<i>usSuspendFilter</i>	USHORT — input Not supported.
<i>usNbrOfClasses</i>	USHORT — input The number of index class identifiers in the list you specify as the value of the <i>pusClassIdList</i> parameter. Specify the value 0 for the <i>usNbrOfClasses</i> parameter to indicate that class is not a criterion for selecting items.
<i>pusClassIdList</i>	PUSHORT — input The pointer to a list of index class identifiers that indicate the items to select for the table of contents. You can specify the value NULL for the <i>pusClassIdList</i> parameter only if you specify the value 0 for the <i>usNbrOfClasses</i> parameter.
<i>pLinkCriteria</i>	PVOID — input Not supported.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the number of items in the table of contents. If no items satisfy the filter, the field contains the value NULL.
<i>ulParam1</i>	Contains a pointer to a buffer with an array of TOCENTRYSTRUCT data structures. If no items satisfy the filter, the field contains the value NULL. For more information on this data structure, see “TOCENTRYSTRUCT (Table of Contents Entry Data Structure)” on page 157. Restriction: Your application must not modify the buffer containing the array of TOCENTRYSTRUCT data structures. If your application needs to update returned information, it must copy this information into its own memory buffer.
<i>ulParam2</i>	Contains the table of contents handle (<i>hTOC</i>). If no items satisfy the filter, the field contains the value NULL.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_ID • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_ITEM_TYPE • SIM_RC_INVALID_PITEMIDITEM_PTR • SIM_RC_INVALID_PITEMIDITEM_VALUE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_PUSCLASSIDLIST_PTR • SIM_RC_INVALID_USITEMTYPE_VALUE • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects: Each time you use this function, you create a new table of contents handle. You can use this handle later with the **SimLibGetTOCData** and **Ip2GetTOCUpdates** functions, to specify which table of contents to process.

Exceptions: The **SimLibGetTOC** function creates a table of contents that shows the current contents of the workbasket or folder. However, the contents of the workbasket or folder might change after you use this function. Use the **Ip2GetTOCUpdates** function to return a list of the changes. Update the TOCENTRYSTRUCT, which includes *usItemStatus*, to indicate changed entries.

Follow-Up Tasks: When you no longer need a table of contents handle, free it by using the **Ip2CloseTOC** function. That function frees both the table of contents handle (*hTOC*) and the data pointed to by the PTOCENTRYSTRUCT pointer.

Example

```
#include "ekdviapi.h"           // Content Manager for iSeries
HSESSION    hSession;         // Session handle
PITEMID     pszItemID;        // Pointer to an item ID
USHORT      usItemType;       // The item type
USHORT      usWipFilter;      // WIP status of search items
```

```

USHORT      usSuspendFilter; // Suspend status of search items
USHORT      usNbrOfClasses; // # of index class identifiers in
                                // pusClassIdList
PUSHORT     pusClassIdList; // Pointer to list of index class IDs
                                // that indicates TOC items.
PVOID       pLinkCriteria;    // Not used
PASYNCCTLSTRUCT pAsyncCtl;    // Pointer to asynchronous control block.
RCSTRUCT    RC;               // Pointer to return data structure.
USHORT      usNumRows = 0;    // # of returned TOC entries
PTOCENTRYSTRUCT pTocEntry;    // pointer to TOC entries

usItemType   = SIM_ALL;       // Set up item type filter.
usWipFilter   = OIM_ALL;      // Set up Work-In-Process status filter
usSuspendFilter = OIM_ALL;    // Set up suspend status of search items.
usNbrOfClasses = 1;          // Set up index class filter
usClassIdList[0] = NO_INDEX;

u1RC = SimLibGetTOC(
    hSession,                // Handle to a Content Manager for iSeries.
    pfoldid,                 // Pointer to folder or Workbasket ID.
    SIM_ALL,                  // The item type filter.
    NULL,                    // WIP status of search items.
    NULL,                    // Suspend status of search items.
    usNbrOfClasses,          // # of index class IDs in pusClassIdList.
    usClassIdList,           // Pointer to index class identifiers list.
    NULL,                    // Not used; link criteria
    NULL,                    // asynch not supported
    &RC                       // pointer to return struct
);
if (u1RC == SIM_RC_OK) {
    hTOC      = (HTOC)RC.ulParam2; // TOC handle
    usNumRows = RC.usParam;        // # of returned toc entries
    pTocEntry = RC.ulParam1;      // pointer to TOC entries.
}

/*****
/* ... Call other Content Manager for iSeries by using the ... */
/* ... session handle obtained by calling SimLibLogon ... */
*****/

u1RC = Ip2CloseTOC(
    hSession,                // Handle to a Content Manager for iSeries
    hTOC,                    // TOC Handle from SimLibGetTOC
    NULL,                    // by NULL, asynchronous call made
    &RC                       // pointer to return struct
);
if (u1RC == SIM_RC_OK) {
    /* Ip2CloseTOC released all resource associated with hTOC */
}

```

Related Functions

- Ip2CloseTOC
- Ip2GetTOCUpdates
- Ip2TOCCount
- Ip2TOCStatus
- SimLibGetTOCData

SimLibGetTOCData (Get a Snapshot of Attributes for a Group of Items)

Format

```
SimLibGetTOCData( hSession, pTOCEntries, ulEntryCount, fDataOptions,
pAsyncCtl, pRC )
```

Purpose

Use the **SimLibGetTOCData** function to return a copy of the attributes associated with a group of documents or folders.

Your application can substitute this function for a series of calls to the **SimLibGetItemSnapshot** function.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pTOCEntries</i>	PTOCENTRYSTRUCT — input The pointer to an array of TOCENTRYSTRUCT data structures that identify the items for which you want a copy of the attributes. For more information on this data structure, see “TOCENTRYSTRUCT (Table of Contents Entry Data Structure)” on page 157.
<i>ulEntryCount</i>	ULONG — input The number of entries in the TOCENTRYSTRUCT array. Because each entry can result in a large amount of data, you should limit the number of entries.
<i>fDataOptions</i>	BITS — input The type of data to return for each item. You must specify at least one value for this parameter. The following are valid values. You can use a bit-wise inclusive OR operator () to combine them. SIM_TOC_SNAPSHOT_SYSTEM_ATTR Returns the system-defined attribute values for the documents or folders. SIM_TOC_SNAPSHOT_USER_ATTR Returns the user-defined attribute values for the documents or folders. SIM_TOC_SNAPSHOT_WORK_ATTR Returns the work management information for the documents or folders. SIM_TOC_SNAPSHOT_ALL Returns the information specified in all the other values.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 1, to indicate that *ulParam1* contains a pointer to a data area. If an error occurs, *usParam* contains the value 0.

ulParam1 Contains a pointer to an array of SNAPSHOTSTRUCT data structures that provide the returned information.

If *usParam* contains the value 0, *ulParam1* contains the array index of the TOCENTRYSTRUCT element that was in error. For some error conditions, the function can identify the item that failed. If not, this field contains *SIM_TOC_MAX_ENTRY_COUNT*.

ulParam2 Contains a count of the items in the returned array. This count matches the value in the *ulEntryCount* parameter.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_BUFFER_NULL
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_INDEX_CLASS
- SIM_RC_INVALID_ITEM_ID
- SIM_RC_INVALID_ITEM_TYPE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_READATTRIND
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- This function retrieves data for any group of folders or documents that you identify properly. It retrieves information for the items returned by the **SimLibGetTOC** function, processing an entire list with one function call. Retrieving work management information takes significantly more time than retrieving attributes.
- Effects vary with the bit values you specify in the *fDataOptions* parameter:
 - If you specify *SIM_TOC_SNAPSHOT_SYSTEM_ATTR* to return system-defined attributes, you always get data if the item is a valid document or folder.
 - If you specify *SIM_TOC_SNAPSHOT_WORK_ATTR* but the item is not in a workbasket, you get a successful return code but the *WMSNAPSHOTSTRUCT* data structure is null.
 - If you specify 0 or an invalid combination of bit values, the function returns *SIM_RC_INVALID_DATA_OPTIONS*.
- All the returned data is in a single memory block. The SNAPSHOTSTRUCT structures appear as an array in the same order as the TOCENTRYSTRUCT structures. The remaining information follows in the same block, referenced by pointers originated in the individual SNAPSHOTSTRUCT structures.

Exceptions:

- The function ignores most of the fields in TOCENTRYSTRUCT. It always uses the item ID field, and it uses the index class when you request user-defined attributes. Therefore, you can use the function to retrieve the item types for a list of folders and documents by preparing a TOCENTRYSTRUCT structure and using only the SIM_TOC_SNAPSHOT_SYSTEM_ATTR value on the *fDataOptions* parameter. The function returns the correct item types in the SNAPSHOTSTRUCT structure.
- Your application might need to use a conversion routine such as an ASCII-to-integer routine to change the character representation of an attribute value into the correct form for the application.

Follow-Up Tasks: After your application has processed the information that the Content Manager for iSeries system returns to the SNAPSHOTSTRUCT data structure, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the pointer to the SNAPSHOTSTRUCT data structure array.

Related Functions

- **SimLibCloseAttr**
- **SimLibFree**
- **SimLibGetItemSnapshot**
- **SimLibGetItemType**
- **SimLibGetTOC**
- **SimLibOpenItemAttr**
- **SimLibReadAttr**

SimLibListClasses (List Index Classes)**Format**

```
SimLibListClasses( hSession, fClassOptions, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibListClasses** function to list all existing index classes in the Content Manager for iSeries database. It lists only the classes for which this user has access and which contain attributes.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>fClassOptions</i>	BITS — input Not supported.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer. Otherwise, this field contains the value 0.
<i>ulParam1</i>	If <i>ulParam2</i> contains a value greater than 0, this field contains a pointer to a buffer. In the buffer, a NAMESTRUCT array provides the index class identifiers and the associated names. For more information on this data structure, see “NAMESTRUCT (Name Data Structure)” on page 149.
<i>ulParam2</i>	Contains the number of fields in the array pointed to by <i>ulParam1</i> .
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects: The name information that this function returns reflects the language defined for the current Content Manager for iSeries session.

Exceptions: This function provides only the identifiers of the index classes in the system that the current user has permission to access. Use the **SimLibGetClassInfo** function to determine the index attributes in an index class.

Follow-Up Tasks: When your application no longer needs the index class identifier list, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

SimLibLogoff (Log Off)

Format

```
SimLibLogoff( hSession, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibLogoff** function to end access to the Content Manager for iSeries operations for a current application.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC

Guidelines for Use

Effects:

- After your application uses this function, any additional Content Manager for iSeries functions fail if they use the same session handle.
- All structures that a Content Manager for iSeries API allocates that are not released using **SimLibFree** are released during logoff.

Example

```
#include <stdio.h>                /* Standard I/O header files */
#include "ekdviapi.h"            /* Content Manager for iSeries */

int main (void) {
    ULONG    ulRC;                /* Return code */
    HSESSION hSession;           /* Session handle */
    PUSERLOGONINFOSTRUCT pUserLogonInfo; /* User logon info struct */
    PSZ      pszDBName="VI400LIB"; /* Pointer to Database name */
    PSZ      pszUserId="QVIADMIN"; /* Pointer to User Id (Name) */
    PSZ      pszPassword="PASSWORD"; /* Pointer to User's Password */
    BITS     fSessionType=1;      /* Product Session Type */
    RCSTRUCT RC;                /* RC data structure */

    /******
    /* Logon to system, and establish a normal session */
    /******
    fSessionType = SIM_SS_NORMAL;
    ulRC = SimLibLogon(
        pszDBName,                // library database
        NULL,                     // not used; library tableset
        pszUserId,                // user ID
        pszPassword,              // user ID password
        NULL,                     // if any, new password
        NULL,                     // not used; proxy ID
        NULL,                     // not used; proxy scope
        fSessionType,            // session access
        NULL,                     // NULL = synchronous call
        &RC                      // pointer to return data struct
    );
    if (ulRC == SIM_RC_OK
        // hSession session handle and user logon info structure
        // returned through RC structure.
```

SimLibLogoff

```
        hSession      = (HSESSION)RC.u1Param1;
        pUserLogonInfo = (PUSERLOGONINFOSTRUCT)RC.u1Param2;
    } else {
        printf("error -SimLibLogon failed with %ld.\n",u1RC);
        exit(1);
    }

    /*****
    /* Call other Content Manager for iSeries APIs by using the */
    /* session handle obtained by calling SimLibLogon          */
    *****/

    /*****
    /* Logoff from system, and end a normal session          */
    *****/
    u1RC = SimLibLogoff(
        hSession,          // Session handle
        NULL,              // not supported
        &RC                // pointer to return data struct
    );
    if (u1RC == SIM_RC_OK) {
        /*****
        /* Logoff success */
        *****/
    } else {
        printf("error - SimLibLogoff failed with %ld\n.",u1RC);
        exit(1);
    }
    return (0);
}
```

Related Functions

- SimLibLogon

SimLibLogon (Log On)

Format

```
SimLibLogon( pszDBName, pszApplicationName, pszUserID, pszPassword,  
pszNewPassword, pszProxyID, pszProxyScope, fSession, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibLogon** function to enable your application to access Content Manager for iSeries operations. Your application must use this function before it can use any other Content Manager for iSeries functions, and it must use the **SimLibLogoff** function when it has finished using Content Manager for iSeries operations.

Parameters

pszDBName PSZ — input

The system name contained in FRNOLINT.TBL.

pszApplicationName

PSZ — input

Not supported.

pszUserID PSZ — input

The NULL-terminated character string that specifies the user ID of the user to log on. Not case sensitive.

<i>pszPassword</i>	PSZ — input The NULL-terminated character string that specifies the password for the user ID. Case sensitivity is based on the iSeries operating system definition in system value QPWDLVL.
<i>pszNewPassword</i>	PSZ — input The NULL-terminated character string that specifies a valid new password for the user ID. Case sensitivity is based on the iSeries operating system definition in system value QPWDLVL. Null to keep existing password.
<i>pszProxyID</i>	PSZ — input Not supported.
<i>pszProxyScope</i>	PSZ — input Not supported.
<i>fSession</i>	BITS — input SIM_SS_NORMAL As part of the logon process, index class and attribute information is retrieved. This improves the performance of subsequent calls. SIM_SS_CONFIG Only the USERLOGONINFOSTRUCT is returned from the server. See “USERLOGONINFOSTRUCT (User Logon Information Structure)” on page 159 for more information on this data structure.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0, to indicate that <i>ulParam1</i> contains a session handle and <i>ulParam2</i> contains a pointer to a buffer.
<i>ulParam1</i>	Contains an <i>hSession</i> parameter or NULL.
<i>ulParam2</i>	Contains a pointer to a USERLOGONINFOSTRUCT data structure. See “USERLOGONINFOSTRUCT (User Logon Information Structure)” on page 159 for more information on this data structure.
<i>ulRC</i>	Contains one of the following return codes. “Guidelines for Use” contains more detail. <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_GRACE_PERIOD_ENDED • SIM_RC_GRACE_PERIOD_OVER_LIMIT

- SIM_RC_INVALID_PASSWORD
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_USERID
- SIM_RC_USERID_UNKNOWN

When the function completes successfully, it returns a value of zero (SIM_RC_OK).

Guidelines for Use

Follow-Up Tasks: After your application gets the information from the USERLOGONINFOSTRUCT data structure, use the **SimLibFree(hSession, (PVOID)ulParam2, pRC)** function to free the memory.

Example

```

#include <stdio.h> /* Standard I/O header files */
#include "ekdviapi.h" /* Content Manager for iSeries */

int main (void) {
    ULONG    ulRC; /* Return code */
    HSESSION hSession; /* Session handle */
    PUSERLOGONINFOSTRUCT pUserLogonInfo; /* User logon info struct*/
    PSZ    pszDBName="VI400LIB"; /* Pointer to Database name */
    PSZ    pszUserId="QVIADMIN"; /* Pointer to User Id (Name) */
    PSZ    pszPassword="PASSWORD"; /* Pointer to User's Password */
    BITS    fSessionType=1; /* Product Session Type */
    RCSTRUCT RC; /* RC's data structure */

    /******
    /* Logon to system, and establish a normal session */
    /******
    fSessionType = SIM_SS_NORMAL;
    ulRC = SimLibLogon(
        pszDBName, /* library database
        NULL, /* not used; library tableset
        pszUserId, /* user ID
        pszPassword, /* user ID password
        NULL, /* if any, new password
        NULL, /* not used; proxy ID
        NULL, /* not used; proxy scope
        fSessionType, /* session access
        NULL, /* not supported
        &RC /* pointer to return data struct
    );
    if (ulRC == SIM_RC_OK ||

        // hSession session handle and user logon info structure
        // returned through RC structure.
        hSession = (HSESSION)RC.ulParam1;
        pUserLogonInfo = (PUSERLOGONINFOSTRUCT)RC.ulParam2;
    } else {
        printf("error -SimLibLogon failed with %ld.\n",ulRC);
        exit(1);
    }

    /******
    /* Call other Content Manager for iSeries APIs by using the */
    /* session handle obtained by calling SimLibLogon */
    /******

    /******
    /* Logoff from system, and end a normal session */
    /******

```



```

u1RC = SimLibLogoff(
    hSession,           // Session handle
    NULL,              // NULL indicates synchronous call
    &RC                // pointer to return data struct
);
if (u1RC == SIM_RC_OK) {
    /******
    /* Logoff success */
    /******
} else {
    printf("error - SimLibLogoff failed with %ld\n.",u1RC);
    exit(1);
}
return (0);
}

```

Related Functions

- SimLibFree
- SimLibLogoff

SimLibOpenItemAttr (Open Item Attributes)

Format

```

SimLibOpenItemAttr( hSession, pszItemID, usClassId, ulAccessLevel, pAsyncCtl,
pRC )

```

Purpose

Use the **SimLibOpenItemAttr** function to provide access to the attributes of a document or folder that you specify. This function opens the item for either read or write access by creating a virtual copy of the attributes associated with that item.

Parameters

<i>hSession</i>	HSESSION — input
	The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemID</i>	PITEMID — input
	The identifier of an item that you want to open to access the attributes. This identifier is the item ID.
<i>usClassId</i>	USHORT — input
	The identifier of an index class.
<i>ulAccessLevel</i>	ULONG — input
	The item access mode. The value of this parameter indicates the access mode for locking the item. The valid values are:
	SIM_ACCESS_READ_WRITE
	Locks the item. Use of this value causes the function to fail if another process has the item locked.
	SIM_ACCESS_SHARED_READ
	Opens the item for read access only. Use of this value opens the item whether or not others have locked it.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input
	Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 0. If the return code is SIM_RC_ITEM_CHECKEDOUT, this field contains the value 1, to indicate that *ulParam1* contains a pointer. If the Content Manager for iSeries system returns any other error, this field contains the value NULL.

ulParam1 Contains an item handle with the data type HITEM, for an open item. If the return code is SIM_RC_ITEM_CHECKEDOUT, this field contains a pointer to a USERACCESSSTRUCT data structure. The data structure contains the user ID of the user who has locked the item.

ulParam2 Returns the index class of the item.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_ASYNC_STARTED
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INUSE
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_INDEX_CLASS
- SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE
- SIM_RC_INVALID_PITEMIDITEM_PTR
- SIM_RC_INVALID_PITEMIDITEM_VALUE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_USACCESSLEVEL_VALUE
- SIM_RC_INVALID_USATTRIBUTEID_VALUE
- SIM_RC_INVALID_USCLASSID_VALUE
- SIM_RC_ITEM_CHECKEDOUT
- SIM_RC_ITEM_NOT_FOUND
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PITEM_NOT_FOLDER_OR_DOCUMENT
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- If your application uses this function with read access, the Content Manager for iSeries system makes a copy of the current attribute values in the database. Concurrent or subsequent access by another user might change those values.
- If your application opens an item for read access while it is open for write access by another application, the values of the item attributes are the same as those currently in the database.
- If you already have the item open for write access, the function returns SIM_RC_INUSE.

- This function returns a handle to the virtual item. This handle, *hItem*, is valid only within the current session. It cannot be transferred to another session. To manipulate the attributes of the item, use the item handle with the **SimLibReadAttr** and **SimLibWriteAttr** functions. To copy the new values permanently, use **SimLibSaveAttr** or **SimLibCloseAttr**.
- **SimLibOpenItemAttr** does not validate if you have SIM_ACCESS_READ_WRITE authority. **SimLibCloseAttr** validates authority when called with SIM_OPT_SAVE.

Exceptions:

- If an item is locked, only the user with the locked item can work with the item. Other users can gain read access only.
- If an item is not locked, all users can gain read access, and the first user with proper authority to request write access gets exclusive update access.
- If another user modifies the attribute values of the item without saving them by using the **SimLibSaveAttr** function, the attribute values you see can be different from the attribute values that the other user sees.

Follow-Up Tasks:

- If you receive the SIM_RC_ITEM_CHECKEDOUT return code and your application no longer needs the user access information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.
- If you receive the SIM_RC_OK return code, use **SimLibCloseAttr** to close the item and release the storage for the item handle. Do not use both the **SimLibFree** and the **SimLibCloseAttr**.

Related Functions

- **SimLibCloseAttr**
- **SimLibReadAttr**
- **SimLibSaveAttr**
- **SimLibWriteAttr**

SimLibOpenObject (Open an Object)

Format

```
SimLibOpenObject( hSession, hObj, ulAccessLevel, ulPriority, fConflict,  
fOpenControl, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibOpenObject** function to prepare an existing object for access by your application. On successful completion, the function returns an object access handle that you can use to access the object.

Parameters

- | | |
|-----------------|---|
| <i>hSession</i> | HSESSION — input
The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information. |
| <i>hObj</i> | HOBJ — input
The pointer to an object handle block in the HOBJ data structure. For more information on the HOBJ structure, see “HOBJ (Handle to Query Stored Object)” on page 143. |

SimLibOpenObject

<i>ulAccessLevel</i>	ULONG — input
	The object access mode. The value of this parameter indicates the access mode for opening the object.
	The Content Manager for iSeries system uses this access state to accept or reject concurrent requests to access an open object. The valid values are:
	SIM_ACCESS_READ_WRITE Opens the object for read access and write access, at the first byte of the object.
	SIM_ACCESS_SHARED_READ Opens the object for read access only, at the first byte of the object.
<i>ulPriority</i>	ULONG — input
	Not supported.
<i>fConflict</i>	BOOL — input
	Not supported.
<i>fOpenControl</i>	BITS — input
	Not supported.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input
	Not supported.
<i>pRC</i>	PRCSTRUCT — input/output
	The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0.
<i>ulParam1</i>	Contains <i>hObjAcc</i> , an HOBJACC object access handle. The value in this field identifies the current instance of the accessed object.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INUSE• SIM_RC_INVALID_ACCESS_CODE• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_OBJECT_HANDLE• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_OBJECT_CHECKEDOUT• SIM_RC_OPEN_FAILED• SIM_RC_OUT_OF_MEMORY• SIM_RC_PRIVILEGE_ERROR• SIM_RC_OBJECT_BEINGPROMOTED

Guidelines for Use

Effects:

- If the function returns the object access handle, this handle identifies the current instance of access to the open object. This handle is different from the handle normally used to reference the stored object. Use the object access handle (*hObjAcc*), not the object handle (*hObj*), with the following functions:
 - **SimLibCloseObject**
 - **SimLibReadObject**
 - **SimLibResizeObject**
 - **SimLibSeekObject**
 - **SimLibWriteObject**
- If you try to open an object for write access and another user has the item locked, the function returns SIM_RC_OBJECT_CHECKEDOUT but does not return the ID of the user who locked the item. You can use the **SimLibGetItemInfo** function to get the user ID.

Example

SimLibLogon...

```

#include <stdio.h>                /* Standard I/O header files    */
#include <string.h>              /* Standard string header file  */
#include "ekdviapi.h"           /* Content Manager for iSeries  */

main()
{
    HSESSION hSession ;          // from logon
    HOBJ hObj;
    UCHAR ulAccessLevel = SIM_ACCESS_SHARED_READ;
    UCHAR ulPriority = 0;        // not supported
    BOOL fConflict = 0;         // not supported
    BOOL fOpenControl = 0;      // Not supported
    RCSTRUCT RC;
    PRCSTRUCT pRC = &RC;
    POBJ      pObj;             // Created object handle
    USHORT    sResult;         // get rc back
    HOBJACC   hObjAcc;         // object access handle

    // create hobj
    if(0==( pObj=(POBJ) malloc(sizeof(OBJ)))) {
        return(1);
    }
    ( pObj)->ulStruct = sizeof(OBJ);
    strcpy(( pObj)->szItemID,"DA97220AA.AAA");
    strcpy(( pObj)->chRepType,""); // take default
    ( pObj)->ulPart = 1;
    hObj = pObj;
    /*Call the function*/

    sResult = SimLibOpenObject(
        hSession,
        hObj,
        ulAccessLevel,
        ulPriority,
        fConflict,
        fOpenControl,
        0, // synch
        pRC);

    if (pRC->ulRC == SUCCESS) {
        // ulParam1 is HOBACC when call is successful.
    }
}

```

SimLibOpenObject

```
        hObjAcc = pRC->ulParam1;  
        // Mem containing the HOBJACC struct is freed by SimLibCloseObject.  
    }  
}
```

Related Functions

- SimLibCloseObject
- SimLibReadObject
- SimLibResizeObject
- SimLibSeekObject
- SimLibWriteObject

SimLibOpenObjectByUniqueName (Open an Object By its Unique Name)

Format

```
SimLibOpenObjectByUniqueName( hSession, pszUniqueName, ulAccessLevel,  
ulPriority, fConflict, fOpenControl, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibOpenObjectByUniqueName** function to display a form overlay that was created using IBM ImagePlus Workfolder Application Facility for AS/400.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszUniqueName</i>	PSZ — input The unique name of the item containing the object that you want to access.
<i>ulAccessLevel</i>	ULONG — input The object access mode. The value of this parameter indicates the access mode for opening the object. The Content Manager for iSeries system uses this access state to accept or reject concurrent requests to access an open object. The valid values are: SIM_ACCESS_READ_WRITE Opens the object for read access and write access, at the first byte of the object. SIM_ACCESS_SHARED_READ Opens the object for read access only, at the first byte of the object.
<i>ulPriority</i>	ULONG — input Not supported.
<i>fConflict</i>	BOOL — input Not supported.
<i>fOpenControl</i>	BITS — input

	Not supported.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input
	Not supported.
<i>pRC</i>	PRCSTRUCT — input/output
	The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0.
<i>ulParam1</i>	Contains <i>hObjAcc</i> , an HOBJACC object access handle. The value in this field identifies the current instance of the accessed object.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INUSE • SIM_RC_INVALID_ACCESS_CODE • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_OBJECT_HANDLE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_OBJECT_CHECKEDOUT • SIM_RC_OPEN_FAILED • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- If the function returns the object access handle, this handle identifies the current instance of access to the open object. This handle is different from the handle normally used to reference the stored object. Use the object access handle (*hObjAcc*), with the following functions:
 - **SimLibCloseObject**
 - **SimLibReadObject**
 - **SimLibResizeObject**
 - **SimLibSeekObject**
 - **SimLibWriteObject**
- If you try to open an object for write access and another user has the item locked, the function returns SIM_RC_OBJECT_CHECKEDOUT but does not return the ID of the user who locked the item. You can use the **SimLibGetItemInfo** function to get the user ID.

Related Functions

- **SimLibCloseObject**
- **SimLibReadObject**
- **SimLibResizeObject**
- **SimLibSeekObject**

- SimLibWriteObject

SimLibQueryObject (Query an Object)

Format

```
SimLibQueryObject( hSession, hObj, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibQueryObject** function to get the information associated with the object that you specify, such as its size and its content class and collection name. This function allocates a buffer for an object information structure and then fills this structure with all the information associated with the object. You do not need to open the object to query it.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObj</i>	HOBJ — input The pointer to an object handle block in the HOBJ data structure. This handle specifies the object that you want to query. For more information on the HOBJ structure, see “HOBJ (Handle to Query Stored Object)” on page 143.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a pointer to a buffer where an OBJINFOSTRUCT data structure contains all the information associated with the object. For more information on this data structure, see “OBJINFOSTRUCT (Object Information Structure)” on page 149.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_ASYNC_STARTED• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE• SIM_RC_INVALID_OBJECT_HANDLE• SIM_RC_INVALID_POINTER

- SIM_RC_INVALID_PRC
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PART_NOT_FOUND
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects: This function returns the data in the OBJINFOSTRUCT.

Follow-Up Tasks: After the function gets the object information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

SimLibReadAttr (Read an Attribute)

Format

```
SimLibReadAttr( hSession, hItem, usAttributeId, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibReadAttr** function to return the value of a specific attribute of the open folder or document you specify.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hItem</i>	HITEM — input The handle to a virtual item, the open folder or document for which you want to read an attribute. The SimLibOpenItemAttr function returns this handle. This item can currently be open in either read or write access mode.
<i>usAttributeId</i>	USHORT — input The unique identifier assigned to an attribute.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer. If an error occurs, this field contains the value 0.
<i>ulParam1</i>	Contains a pointer to a buffer in which a null-terminated string is a character representation of the attribute value. If the attribute value is undefined, the value is NULL.
<i>ulParam2</i>	The function does not use this field.

SimLibReadAttr

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HITEM_VALUE
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_USATTRIBUTEID_VALUE
- SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Exceptions:

- Attributes are always returned as a NULL-terminated string.
- Your application might need to use a conversion routine such as an ASCII-to-integer routine to change the character representation of the value into the correct form for the application.
- Use the **SimLibGetAttrInfo** function to get the data types and lengths of attributes. Use the **SimLibGetItemInfo** function and the **SimLibGetClassInfo** function to get the class attributes.

Follow-Up Tasks: When you no longer need the attribute string, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

Related Functions

- **SimLibGetClassInfo**
- **SimLibGetAttrInfo**
- **SimLibGetItemInfo**
- **SimLibOpenItemAttr**

SimLibReadObject (Read an Object)

Format

```
SimLibReadObject( hSession, hObjAcc, pBuffer, ulBytesToRead, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibReadObject** function to transfer the number of bytes you specify from an object into the data buffer of your application. This function lets you manipulate an object as a file. The function begins reading the object at the byte that the object pointer is currently referencing.

Parameters

hSession HSESSION — input
The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

hObjAcc HOBJACC — input
The object access handle to the open object that you want to read into the data buffer of your application. The value of this parameter identifies the current instance of the accessed object.

pBuffer PHBUF — input

	The data buffer pointer. The value of this parameter represents a pointer to the first byte of the buffer returning the read object data.
<i>ulBytesToRead</i>	ULONG — input The number of bytes to read. The value of this parameter specifies the maximum number of bytes to read from the object during the transfer operation.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a pointer to the byte immediately after the last byte written to the buffer. Normally, this is the address of the buffer plus the number of bytes read.
<i>ulParam2</i>	Contains the actual number of bytes read.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_BUFFER_PTR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_OBJECT_ACCESS_HANDLE • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_OUT_OF_MEMORY • SIM_RC_READ_PAST_EOF

Guidelines for Use

Preparation: Before you can read the object, you must open it and obtain an object access handle.

Effects: After successful completion of the function, the object pointer references the byte immediately following the data that was read.

Exceptions: If the number of bytes that you specify to be read is more than the number of bytes in the object, the function transfers fewer bytes than you specify.

Related Functions

- SimLibCloseObject
- SimLibOpenObject
- SimLibSeekObject

SimLibRemoveFolderItem (Remove an Item from a Folder)

Format

```
SimLibRemoveFolderItem( hSession, pszFolderID, pszItemID, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibRemoveFolderItem** function to remove a document or a folder item from a folder. This function removes the reference to the item from the table of contents of the specified folder. You need not open the folder to use the function, but the folder must not be locked by another user.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszFolderID</i>	PITEMID — input The identifier of a folder from which you want to remove an item. This identifier is the item ID of the folder.
<i>pszItemID</i>	PITEMID — input The identifier of an item to remove from the folder. This identifier is the item ID of a document or a folder item.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	If the return code is SIM_RC_PARENT_CHECKEDOUT, this field contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains the value NULL. If the return code is SIM_RC_PARENT_CHECKEDOUT, this field contains a pointer to a USERACCESSSTRUCT data structure. The structure contains the user ID of the user who has locked the folder.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_PITEMIDFOLDER_PTR • SIM_RC_INVALID_PITEMIDFOLDER_VALUE • SIM_RC_INVALID_PITEMIDITEM_PTR

- SIM_RC_INVALID_PITEMIDITEM_VALUE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PARENT_CHECKEDOUT
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- If the folder is locked by another user, you cannot remove an item from it. Instead, the function returns the user ID of the user who has locked the folder. If you have locked the folder, you can remove items from it.

Exceptions:

- This function does not automatically update a temporary copy of the table of contents for a folder. Your application must use either the **Ip2GetTOCUpdates** function or the **SimLibGetTOC** function to update the table of contents of this folder.
- You can remove an item that you or someone else has locked. Only the status of the parent folder is examined.

Follow-Up Tasks: After your application no longer needs the user access information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer containing the USERACCESSSTRUCT data structure.

Related Functions

- **Ip2GetTOCUpdates**
- **SimLibAddFolderItem**
- **SimLibDeleteItem**
- **SimLibFree**
- **SimLibGetTOC**

SimLibResizeObject (Resize an Object)

Format

```
SimLibResizeObject( hSession, hObjAcc, ulSize, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibResizeObject** function to change the size, in bytes, of an object to a new size that you specify.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObjAcc</i>	HOBJACC — input The object access handle to the object that you want to resize. The value of this parameter identifies the current instance of the accessed object.
<i>ulSize</i>	ULONG — input

SimLibResizeObject

The new object size. To truncate the object file beginning at the current position of the object pointer, and including that byte, specify the value 0. To truncate the file to a specific byte size, specify that byte size.

pAsyncCtl PASYNCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam The function does not use this field.

ulParam1 The function does not use this field.

ulParam2 The function does not use this field.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_OBJECT_ACCESS_HANDLE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_NO_WRITE_ACCESS
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_RESIZE_FAILED
- SIM_RC_SEEK_ERROR

Guidelines for Use

Preparation: Before you use this function to resize an object, the object must be open for SIM_ACCESS_READ_WRITE access.

Effects:

- The object file pointer is set to the end of the object at the completion of this function.
- Use this function when you want to replace an object with one that is smaller than the original. Use **SimLibWriteObject** and then **SimLibResizeObject** to truncate at the end of the new data.

Exceptions: To increase the size of an object, you should use the **SimLibWriteObject** function to append data to the object and increase its size at the same time.

Related Functions

- **SimLibWriteObject**

SimLibSaveAttr (Save an Attribute)

Format

```
SimLibSaveAttr( hSession, hItem, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibSaveAttr** function to save the attributes of a virtual item permanently. This function saves work that is in process on a virtual item without closing the item or releasing access rights.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hItem</i>	HITEM — input The handle to a virtual item. The SimLibOpenItemAttr function returns this handle.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_ATTRIBUTES_NOT_MODIFIED • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HITEM_VALUE • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_PASSED_ATTR_DATA • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_USCLASSID_VALUE • SIM_RC_NO_WRITE_ACCESS • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR • SIM_RC_REQUIRED_ATTRIBUTE_MISSING

Guidelines for Use

Effects:

- If a virtual item is open for write access and modified, this function copies the attributes of the virtual item over the attributes in the database.
- If the index class is changed, this function saves a new set of user-defined attributes in the new index class and deletes the old attributes.

Related Functions

- SimLibOpenItemAttr

SimLibSearch (Search)

Format

```
SimLibSearch( hSession, pszItemFilter, pLinkCriteria, usStatDyn, usTypeFilter,
fWipFilter, usSuspendFilter, usIndexClass, usNumCriteria, pCriteria,
ulMemListRequest, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibSearch** function to locate items in the database that match the user-defined attribute values you specify.

This function returns items that match the search criteria to the user. If you specify an index class, you can search on values of user-defined attributes within the index class. If you do not specify an index class, this function searches only index classes that contain all specified user-defined attributes. For example, in a request to search all index classes for “account number” equal to 12345, the search is limited to those index classes that include “account number” as a user-defined attribute. You can specify multiple combinations of index classes and attributes.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszItemFilter</i>	PITEMID — input Not supported.
<i>pLinkCriteria</i>	PVOID — input Not supported.
<i>usStatDyn</i>	USHORT — input Not supported.
<i>usTypeFilter</i>	USHORT — input The type of items to search for. The valid values are: SIM_DOCUMENT Searches for documents. SIM_FOLDER Searches for folders. SIM_FOLDER_DOC Searches for both folders and documents.

<i>fWipFilter</i>	BITS — input Not supported.
<i>usSuspendFilter</i>	USHORT — input Not supported.
<i>usIndexClass</i>	USHORT — input Not supported.
<i>usNumCriteria</i>	USHORT — input The number of fields in the <i>pCriteria</i> array.
<i>pCriteria</i>	PLIBSEARCHCRITERIASTRUCT — input The pointer to an array specifying the search criteria for each view you want to search. <i>pCriteria</i> must point to an array of at least one field. For more information on the LIBSEARCHCRITERIASTRUCT structure, see “LIBSEARCHCRITERIASTRUCT (Search Criteria Information Structure)” on page 147.
<i>ulMemListRequest</i>	BOOL — input This parameter controls how the search results are returned or which attribute values are returned. The valid values are: SIM_SEARCH_MEMLIST Returns the search results in a memory buffer. SIM_SEARCH_MEMLIST_ONE Not supported. SIM_SEARCH_USER_ATTR Returns the item IDs and user attributes for the item in a memory buffer. SIM_SEARCH_USER_SYSTEM_ATTR Returns the item IDs, user attributes, and system attributes in a memory buffer.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer to a buffer. If nothing matches the input search criteria, this field contains the value 0.
<i>ulParam1</i>	If you set the <i>ulMemListRequest</i> parameter to

SIM_SEARCH_MEMLIST, this field contains a PITEMID pointer to a buffer. In the buffer, an array provides document and folder item IDs that match the search criteria.

If you set the *ulMemListRequest* parameter to SIM_SEARCH_USER_ATTR or SIM_SEARCH_USER_SYSTEM_ATTR, this field contains a pointer to an array of SNAPSHOTSTRUCTs containing the attribute data for items that meet the search criteria.

ulParam2 Contains the number of items that match the criteria (the number of fields in the array referenced by *ulParam1*). The values in the *ulReturnLimit* field of the LIBSEARCHCRITERIASTRUCT structures limit this number.

If nothing matches the search criteria, this field contains the value 0.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_ATTR_NOT_IN_VIEW
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_FSEARCH
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_INDEX_CLASS
- SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE
- SIM_RC_INVALID_PATTRIBUTELIST_VALUE
- SIM_RC_INVALID_PITEMIDFOLDER_VALUE
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_SEARCH_STRING
- SIM_RC_INVALID_USATTRIBUTEID_VALUE
- SIM_RC_INVALID_USITEMTYPE_VALUE
- SIM_RC_INVALID_VIEWID
- SIM_RC_NO_SEARCH_CRITERIA
- SIM_RC_NO_SEARCH_VIEWS
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

See Appendix A, "Guidelines for Search Expressions," on page 291.

Effects:

- If nothing matches the input search criteria, the function returns a successful return code and the *usParam*, *ulParam1*, and *ulParam2* fields all contain the value NULL.
- Specifying very explicit search criteria can narrow the number of items returned by the search. Alternatively, specifying very general search criteria might degrade the performance of the search.
- If you specify an all index class search, the function automatically searches only index classes that contain those attributes specified in the expression.

Follow-Up Tasks: If you set the *ulMemListRequest* parameter to SIM_SEARCH_MEMLIST, after the function gets the search results information, use **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) to free the buffer.

SimLibSeekObject (Seek an Object)

Format

```
SimLibSeekObject( hSession, hObjAcc, ulOrigin, lOffset, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibSeekObject** function to adjust the object pointer to reference a new position that you define. The next data transfer operation for the object begins at this new position. Use this function to position the pointer before you change an object. This function lets you manipulate an object as a file.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObjAcc</i>	HOBJACC — input The object access handle to the object in which you want to adjust the object pointer. The value of this parameter identifies the current instance of the accessed object. The SimLibOpenObject function returns this handle.
<i>ulOrigin</i>	ULONG — input The pointer origin index. The value of this parameter indicates the initial position of the object pointer. The valid values are: SIM_POS_BEGIN Indicates the beginning of the object. SIM_POS_CURRENT Indicates the current pointer position. SIM_POS_END Indicates the byte following the end of the object.
<i>lOffset</i>	LONG — input The byte offset from the origin. The value of this parameter specifies the position in the object for the adjusted object pointer to reference. Specify the value in relation to the position you specify as the value of the <i>ulOrigin</i> parameter. This value can be either a negative or a positive byte count.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0.
----------------	-----------------------

SimLibSeekObject

<i>ulParam1</i>	Contains <i>ulOffset</i> , the current offset, which has the data type ULONG. This value indicates the offset, in bytes, from the beginning of the object. If the current position is at the beginning of the object, this value is 0.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_OBJECT_ACCESS_HANDLE• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_INVALID_SEEK_OFFSET• SIM_RC_INVALID_SEEK_ORIGIN• SIM_RC_OUT_OF_MEMORY• SIM_RC_RESIZE_FAILED• SIM_RC_SEEK_ERROR

Guidelines for Use

Preparation: You must have opened the object and obtained an *hObjAcc* by calling **SimLibOpenObject** before you can call the **SimLibSeekObject** function.

Effects: You can adjust the object pointer to reference a position beyond the end of the object. However, any attempt to reference a position before the beginning of the object returns error code SIM_RC_INVALID_SEEK_OFFSET.

Related Functions

- SimLibOpenObject

SimLibStageObject (Stage an Object)

Format

```
SimLibStageObject( hSession, hObj, ulPriority, fStageControl, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibStageObject** function to retrieve an object from secondary storage to iSeries DASD.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObj</i>	HOBJ — input The pointer to an object handle block in the HOBJ data structure. For more information on the HOBJ structure, see “HOBJ (Handle to Query Stored Object)” on page 143.
<i>ulPriority</i>	ULONG — input The priority value, which specifies the servicing priority for the object. The valid values are:

SIM_PRI_IMMEDIATE

Attempt to interactively retrieve the object.

SIM_PRI_BACKGROUND

Generate a retrieve request for the object.

fStageContro

BITS — input

Control option bits for staging the object. The valid value is:

SIM_PREFETCH

To prefetch to object server.

pAsyncCtl

PASYNCTLSTRUCT — input

Not supported.

pRC

PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam The function does not use this field.

ulParam1 The function does not use this field.

ulParam2 The function does not use this field.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_FOPTIONS
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_HSYNC
- SIM_RC_INVALID_OBJECT_HANDLE
- SIM_RC_INVALID_PRC
- SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Preparation: If you are using this API to generate retrieve requests, the optical retrieve processor must be started and running to actually retrieve the object.

Effects: On successful completion of the function, either a retrieve request will be generated for the object or the object will be interactively retrieved.

Related Functions

- SimLibLogon

SimLibStoreNewObject (Store a New Object in an Existing Item)**Format**

SimLibStoreNewObject(*hSession, hObj, ulConCls, pSMS, pObjBuffer, ulObjSize, lSeqAfterPart, ulAffiliatedType, pAffiliatedData, pAsyncCtl, pRC*)

SimLibStoreNewObject

Purpose

Use the **SimLibStoreNewObject** function to add a new object to an existing item. This is a streamlined version of the **SimLibCatalogObject** function with fewer options and data checks.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObj</i>	HOBJ — input The pointer to an object handle block. For more information on the HOBJ structure, see “HOBJ (Handle to Query Stored Object)” on page 143.
<i>ulConCls</i>	ULONG — input The content class identifier for the object (see Appendix B, “Predefined Content Classes,” on page 295). The value of this parameter tells what kind of data is in the new object. To indicate the undefined content class, specify the value SIM_CC_UNKNOWN for this parameter. However, if you have created an undefined content class, other applications cannot use Content Manager for iSeries content class services to determine how to manipulate the contents of the objects you store.
<i>pSMS</i>	PSMS — input Pointer to a system-managed storage (SMS) structure for an object. This structure uses only <i>szCollectionName</i> .
<i>pObjBuffer</i>	PVOID — input The pointer to a memory buffer containing the object data.
<i>ulObjSize</i>	ULONG — input The total size, in bytes, of the object.
<i>lSeqAfterPart</i>	LONG — input Not supported.
<i>ulAffiliatedType</i>	LONG — input The type of affiliated object to store. The defined values are: SIM_ANNOTATION Stores an annotation associated with a folder or a document. SIM_BASE Stores a base object such as a MO:DCA or TIFF file, that is not an annotation, note, or event associated with a folder or document. SIM_EVENT Stores an event associated with a folder or a document. SIM_MGDS Stores an MGDS (machine-generated data stream) associated with a folder or a document.

SIM_NOTE

Stores a note associated with a folder or a document.

pAffiliatedData PVOID — input

The pointer to a data structure of the type ANNOTATIONSTRUCT. If the *ulAffiliatedType* parameter contains the value SIM_ANNOTATION, *pAffiliatedData* points to this structure, which contains additional data affiliated with the object. Otherwise, the Content Manager for iSeries system ignores this parameter. For more information on the ANNOTATIONSTRUCT structure, see “ANNOTATIONSTRUCT (Annotation Information Structure)” on page 134.

pAsyncCtl PASYNCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam The function does not use this field.

ulParam1 The function does not use this field.

ulParam2 The function does not use this field.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_ANNOTATIONSTRUCT_PTR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_SMS_PTR
- SIM_RC_OUT_OF_MEMORY
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use**Preparation:**

- To get the supported values for the *ulConCls* parameter, use the **Ip2ListContentClasses** function.
- If 0 is specified for the part number, the next sequential part number is created. If part number is nonzero, that part number is used if it does not already exist. If it does exist, the first available number is returned. Part number 1 is typically a base part. This API lets you create part number 2 – for example, a note – before creating part number 1.

Exceptions: The Content Manager for iSeries system does not validate the content class parameter as a defined, known content class.

Related Functions

- Ip2ListContentClasses
- SimLibCatalogObject

SimLibWriteAttr (Write an Attribute)

Format

```
SimLibWriteAttr( hSession, hItem, usAttributeId, pszAttributeValue, pAsyncCtl, pRC )
```

Purpose

Use the **SimLibWriteAttr** function to assign a value to an attribute associated with an open item. You can only modify a user-defined attribute.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hItem</i>	HITEM — input The handle to a virtual item. The SimLibOpenItemAttr function returns this handle. To use the SimLibWriteAttr function, the item must currently be open in write access mode.
<i>usAttributeId</i>	USHORT — input The unique identifier assigned to an attribute.
<i>pszAttributeValue</i>	PSZ — input A null-terminated character string containing the value of an attribute. This string contains the value you assign to the attribute you specify in the <i>usAttributeId</i> parameter.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_ATTRIBUTE_READ_ONLY

- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HITEM_VALUE
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_PASSED_ATTRIBUTE_DATA
- SIM_RC_INVALID_PATTRIBUTE_PTR
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_INVALID_USATTRIBUTEID_VALUE
- SIM_RC_NO_WRITE_ACCESS
- SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Preparation: Use a conversion routine such as an integer-to-ASCII routine to change numeric data into a character string for this function.

Effects:

- This function copies the value of the *pszAttributeValue* parameter into the virtual item.
- The item must be open for write access or the function returns an error, SIM_RC_NO_WRITE_ACCESS.
- If the function fails, the Content Manager for iSeries system maintains the current attribute value.

Exceptions:

- The **SimLibWriteAttr** function validates only SIM_ATTR_FSTRING data types. It validates these data types by comparing maximum lengths of the attribute data with the Content Manager for iSeries-defined string. The **SimLibCloseAttr** and the **SimLibSaveAttr** functions validate the attribute contents by comparing the data with the data types configured through the **SimLibWriteAttr** function.
- The **SimLibWriteAttr** function changes only the virtual copy in memory. It does not update the permanent database copy of the attribute. Use the **SimLibSaveAttr** or the **SimLibCloseAttr** function to make the modifications permanent.

Related Functions

- SimLibCloseAttr
- SimLibGetAttrInfo
- SimLibGetClassInfo
- SimLibOpenItemAttr
- SimLibSaveAttr

SimLibWriteObject (Write an Object)

Format

```
SimLibWriteObject( hSession, hObjAcc, pBuffer, ulBytesToWrite, pAsyncCtl,
pRC )
```

Purpose

Use the **SimLibWriteObject** function to transfer the number of bytes you specify from the data buffer of your application to an open object. The write operation begins at the byte referenced by the current object pointer.

SimLibWriteObject

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hObjAcc</i>	HOBJACC — input The object access handle to the object that you want to write to. The value of this parameter identifies the current instance of the accessed object.
<i>pBuffer</i>	PHBUF — input The data buffer pointer. The value of this parameter represents a pointer to the first byte of the data to be written to the object.
<i>ulBytesToWrite</i>	ULONG — input The number of bytes to write to the object. The value of this parameter specifies the maximum number of bytes to write to the object during the transfer operation.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0.
<i>ulParam1</i>	Contains the number of bytes actually written.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_BUFFER_PTR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_OBJECT_ACCESS_HANDLE• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_NO_WRITE_ACCESS• SIM_RC_OUT_OF_MEMORY• SIM_RC_RESIZE_FAILED

Guidelines for Use

Preparation:

- Before you can use this function, you must open the object with SIM_ACCESS_READ_WRITE access using one of the following functions:
 - **SimLibOpenObject**
 - **SimLibCreateObject**
 - **SimLibCatalogObject**

- If you are replacing an object with one that is smaller than the original, first truncate the original object to the size of the replacement object using the **SimLibResizeObject** function. Then you can replace the object using the **SimLibWriteObject** function. If the replacement object is larger than the original, resizing first is not necessary.

Effects: On successful completion of the function, the object pointer references the byte immediately following the data that was written.

Example

```
#include <stdio.h>                /* Standard I/O header files    */
#include <string.h>               /* Standard string header file  */
#include "ekdviapi.h"            /* Content Manager for iSeries  */

main()
{
    HSESSION hSession;           // get from logon
    HOBJACC hObjAcc;             // get from catalog, open, or create
    RCSTRUCT RC;
    PRCSTRUCT pRC = &RC;
    USHORT    sResult;           // return codes
    CHAR    pBuffer[4096];       // buffer
    ULONG    ulBytesToWrite = 2048;

    /* fill buffer */

    /*Call the function*/

    sResult = SimLibWriteObject(
        hSession,
        hObjAcc,
        pBuffer,
        ulBytesToWrite,
        pAsyncCtl,
        pRC);

    if ((pRC->ulRC == SIM_RC_OK) && (ulBytesToWrite != pRC->ulParam1))
        printf("not all the bytes got written");
}

```

Related Functions

- **SimLibCatalogObject**
- **SimLibCreateObject**
- **SimLibOpenObject**
- **SimLibResizeObject**
- **SimLibWriteObject**

SimWmActivateWorkPackage (Activate a Work Package)

Format

```
SimWmActivateWorkPackage( hSession, ulWorkPackageID, ulInstanceID,  
pAsyncCtl, pRC )
```

Purpose

Use the **SimWmActivateWorkPackage** function to release a suspended work package.

SimWmActivateWorkPackage

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_PRC

Related Functions

- SimWmSuspendWorkPackage

SimWmBeginProcess (Start a Work Package on a Pre-defined Process)

Format

```
SimWmBeginProcess( hSession, pszProcessID, pszRouteName,  
pszWorkPackageDesc, ulNumVariables, pVariableList, usPriority, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmBeginProcess** function to create a work package containing the item and start the work package on a predefined process.

Parameters

<i>hSession</i>	HSESSION — input
-----------------	------------------

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

<i>pszProcessID</i>	PSZ — input The identifier of the process.
<i>pszRouteName</i>	PSZ — input Pointer to the name of the initial route within the process. If the pointer is NULL, the default route within the specified process is used.
<i>pszWorkPackageDesc</i>	PSZ — input The NULL-terminated character string that specifies the work package description. It can be used as a comment about the task or as information the application uses as a key to an application database for more details about the work.
<i>ulNumVariables</i>	ULONG — input Number of entries in the variable array. Maximum number of entries that can be specified is two. This field is ignored if the array <i>pVariableList</i> pointer is NULL.
<i>pVariableList</i>	PWMVARSTRUCT — input Pointer to an array of WMVARSTRUCT structures containing the variable identifiers and values for work management variables. Valid variable names are: SIMWM_ITEMID The valid value for SIMWM_ITEMID is the item ID of a document or folder. SIMWM_INDEX_CLASS The valid value for SIMWM_INDEX_CLASS is an index class identifier.
<i>usPriority</i>	USHORT — input Priority of the work to be performed. The priority affects the work sequencing of the work package. A larger number is a higher priority. Use a priority of zero to request the default priority.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Always zero.
<i>ulParam1</i>	Contains the work package ID.

SimWmBeginProcess

<i>ulParam2</i>	Contains the work package instance.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• OIM_WB_FULL• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_INDEX_CLASS• SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_INVALID_PROCESS_NAME• SIM_RC_OUT_OF_MEMORY• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Preparation: To associate a work package to an item in an index class, specify variables, SIMWM_INDEX_CLASS and SIMWM_ITEMID. The *pVariableList* parameter can be NULL to reflect a work package with no direct database references. If *pVariableList* is not specified, the calling application is responsible for associating the work package ID to the object.

If the route name is not specified, the work package is routed to the first route in the specified predefined process.

Exceptions: When you use **SimWmBeginProcess** to start a work package on a process, the workbasket overload limit is ignored, meaning that the work package is always added to the workbasket. A return code of OIM_WB_FULL is returned, however, to indicate that the work package was placed in a workbasket whose overload limit has been reached.

SimWmChangeVariables (Change Variable Values for a Work Package)

Format

```
SimWmChangeVariables( hSession, ulWorkPackageID, ulInstanceID,  
ulNumVariables, pVariableList, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmChangeVariables** function to create new variables that are associated with a work package, or to update variables that already exist.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>ulWorkPackageID</i>	ULONG — input

	Identifier of the work package.
<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>ulNumVariables</i>	ULONG — input Number of entries in the variable array.
<i>pVariableList</i>	PWMVARSTRUCT — input Pointer to an array of WMVARSTRUCT structures containing the variable identifiers and values for work management variables.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_WM_VARIABLE

Guidelines for Use

Preparation: The pre-defined variable SIMWM_ACTION (*ACTION) is used by the IBM Content Manager for iSeries client to identify the last action selected by a user. The value assigned to this variable is based on the action list definition.

Exceptions: The variables SIMWM_ITEMID (*ITEMID) and SIMWM_INDEX_CLASS (*INDEXCLASS) are reserved for internal use and may not be created or changed using the **SimWmChangeVariables** function.

Related Functions

- **SimWmQueryVariables**

SimWmCreateWorkPackage (Create a Work Package)

Format

```
SimWmCreateWorkPackage( hSession, pszWorkPackageDesc, ulNumVariables,
pVariableList, usWorkPriority, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmCreateWorkPackage** function to create a new work package that an application can use for ad hoc work control. This allows the application to route a work package containing a folder or document through one or more workbaskets without the requirement for a pre-defined process.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszWorkPackageDesc</i>	PSZ — input Pointer to a description of the work package. It can be used as a comment about the task or as information the application uses as a key to an application database for more details about the work.
<i>ulNumVariables</i>	ULONG — input Number of entries in the variable array. This field is ignored if the array <i>pVariableList</i> pointer is NULL.
<i>pVariableList</i>	PWMVARSTRUCT — input Pointer to an array of WMVARSTRUCT structures containing the variable identifiers and values for work management variables. The parameter can be NULL to reflect a work package with no direct database references or a work package that an application associates to an object. To associate a work package to an item in an index class, include the variables SIMWM_INDEX_CLASS and SIMWM_ITEMID.
<i>usWorkPriority</i>	USHORT — input Priority of the work to be performed. The priority affects the work sequencing of the work package at the workbasket. A larger number is a higher priority. Use a priority of zero to request the default priority.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Always zero.
<i>ulParam1</i>	Contains the work package ID.
<i>ulParam2</i>	Contains the work package instance.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_INDEX_CLASS • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Preparation: You can specify variables to associate a work package with a specific library item. If *pVariableList* is not specified, the calling application is responsible for associating the work package ID to the object that is being processed. If it is specified, then the work management interface always returns the data to the application whenever the work package ID is referenced in an API. For example, when the calling application gets the next work package from a workbasket, the item ID would also be returned.

Effects: A new work package is created.

Follow-Up Tasks: **SimWmRouteWorkPackage** should be called to route the work package to a workbasket.

Related Functions

- **SimWmRouteWorkPackage**

SimWmEndCollectionPoint (Force a Work Package Out of a Collection Point)**Format**

```
SimWmEndCollectionPoint( hSession, ulWorkPackageID, ulInstanceID,
pAsyncCtl, pRC )
```

Purpose

Use the **SimWmEndCollectionPoint** function to force a work package out of a collection point.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package that represents the work being done, such as the document being routed.

SimWmEndCollectionPoint

<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>pAsyncCtl</i>	PASYNCCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• OIM_ITEM_NOT_SUSPENDED• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_PRC• SIM_RC_PRIVILEGE_ERROR

SimWmEndProcess (End a Work Package on a Process)

Format

```
SimWmEndProcess( hSession, ulWorkPackageID, ulInstanceID, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmEndProcess** function to force an end to an active work package. It removes the work package from workbaskets.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input If only one instance exists, this parameter is ignored.
<i>pAsyncCtl</i>	PASYNCCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam The function does not use this field.
ulParam1 The function does not use this field.
ulParam2 The function does not use this field.
ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_PRC
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- If the work package instance field is zero, it is assumed that the the process is being ended; otherwise, the route is ended. If the work package is ended on a process, all instances of the work package are ended.
- To end a work package on an ad hoc route, specify only the work package ID.

Related Functions

- SimWmCreateWorkPackage
- SimWmGetWorkPackage

SimWmGetActionListInfo (Get Action List Information)

Format

```
SimWmGetActionListInfo( hSession, pszActionListName, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmGetActionListInfo** function to obtain the detail information associated with an action list.

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

pszActionListName
PSZ — Input

The pointer to the name of the action list.

pAsyncCtl PASYNCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

SimWmGetActionListInfo

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a pointer to a WMACTIONLISTINFOSTRUCT data structure that provides the action list information. See “WMACTIONLISTINFOSTRUCT (Action List Data Structure)” on page 161 for additional information.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC

Guidelines for Use

Follow-Up Task: When your application no longer needs the WMACTIONLISTINFOSTRUCT data, use the **SimLibFree** function to free the buffer containing the structure.

SimWmGetProcessInfo (Get Information About a Process)

Format

```
SimWmGetProcessInfo(hSession, pszProcessID, fGetProcessInfo, pAsyncCtl, pRC)
```

Purpose

Use the **SimWmGetProcessInfo** function to return detailed information for a specific process defined in the system. This function returns workbaskets and/or collection points associated with a specific process.

Parameters

<i>hSession</i>	HFSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszProcessID</i>	PSZ — input Pointer to the process identifier.
<i>fGetProcessInfo</i>	BITS — input Flag bits that select what information to return about the process. You can use the bitwise inclusive OR operator (!) to combine them.

SIMWWM_PROCESS_WORKBASKETS

Returns information about all workbaskets associated with the specified process.

SIMWWM_PROCESS_COLLECTION_POINTS

Returns information about all collection points associated with the specified process.

SIMWWM_PROCESS_ALL_LOCATIONS

Returns workbasket and collection point information associated with the specified process.

SIMWWM_PROCESS_COUNT

Returns the number of workbaskets and collection points associated with the specified process.

pAsyncCtl PASYNCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

- usParam* Contains the value 1 to indicate that *ulParam1* contains a pointer.
- ulParam1* Contains a pointer to a buffer where a WMPROCESSINFOSTUCT data structure provides the process definition information. For more information on this data structure, see “WMPROCESSINFOSTRUCT (Process Information Data Structure)” on page 163.
- ulParam2* Contains the number of locations. This value is dependent on the setting of *fGetProcessInfo*.
- ulRC* Contains one of the following return codes:
 - SIM_RC_OK
 - SIM_RC_COMPLETION_ERROR
 - SIM_RC_ERROR_READING_FROM_FILE
 - SIM_RC_FILE_NOT_FOUND
 - SIM_RC_INVALID_GETPROCESOPTIONS
 - SIM_RC_INVALID_HSESSION
 - SIM_RC_INVALID_POINTER
 - SIM_RC_INVALID_PRC
 - SIM_RC_ITEM_NOT_FOUND
 - SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the process information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

Related functions:

- SimWmListProcesses

SimWmGetWorkBasketInfo (Get Information about a Workbasket)

Format

```
SimWmGetWorkBasketInfo( hSession, pszWorkBasketID, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmGetWorkBasketInfo** function to return information about the workbasket you specify.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszWorkBasketID</i>	PSZ — input Pointer to the workbasket identifier.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a pointer to a buffer where a WORKBASKETINFOSTRUCT data structure provides detailed information about the specified workbasket. For more information on this data structure, see “WORKBASKETINFOSTRUCT (Workbasket Information Data Structure)” on page 168.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• OIM_INVALID_PITEMIDWB_PTR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_ITEM_ID• SIM_RC_INVALID_PRC• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the WORKBASKETINFOSTRUCT data, use **SimLibFree** to free the buffer.

Related Functions

- **SimWmListWorkBaskets**

SimWmGetWorkPackage (Get the Next Work Package from a Workbasket)

Format

```
SimWmGetWorkPackage( hSession, pszWorkBasketID, ulWorkOrder,  
ulWorkPackageID, ulInstanceID, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmGetWorkPackage** function to get (open) a work package that is currently in a workbasket. The work package that is queued at the specified workbasket is then not available to other applications. This function can get a specific work package or the next work package currently available in the specified workbasket based on work order.

Parameters

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

pszWorkBasketID

PSZ — input

Pointer to the workbasket identifier.

ulWorkOrder ULONG — input

Order used for selecting an entry from the workbasket. The valid values are:

NULL The server determines the work order and returns the first available work package, or returns the requested work package.

SIMWM_ORDER_FIFO

Make selection based on first in, first out (FIFO) order to return first available work package.

SIMWM_ORDER_LIFO

Make selection based on last in, first out (LIFO) order to return first available work package.

SIMWM_ORDER_PRIORITY

Make selection based on the work package priority to return first available work package.

SIMWM_ORDER_SYSTEM_NEXT

The server determines the work order and returns the next available work package.

SimWmGetWorkPackage

SIMWWM_ORDER_FIFO_NEXT

Make selection for the next available work package based on first in, first out (FIFO) order.

SIMWWM_ORDER_LIFO_NEXT

Make selection for the next available work package based on last in, first out (LIFO) order.

SIMWWM_ORDER_PRIORITY_NEXT

Make selection for the next available work package based on the work package priority.

ulWorkPackageID

ULONG — input

Identifier of the work package that represents the work being done, such as the document being routed. Specify zero to retrieve the first work package. If a work package ID is specified, that work package or the next available work package is retrieved, depending on the value specified in *ulWorkOrder*.

ulInstanceID

ULONG — input

Identifier of the work package instance that distinguishes one parallel path from another within the process.

pAsyncCtl

PASYNCTLSTRUCT — input

Not supported.

pRC

PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer to a data area.
<i>ulParam1</i>	Contains a pointer to a SNAPSHOTSTRUCT data structure that provides the returned item and associated work management information.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• OIM_INVALID_PITEMIDWB_PTR• SIM_RC_COMPLETION_ERROR• SIM_RC_EMPTY_WORKBASKET• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_PRC• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- If the work package ID is not specified, this function will retrieve the first available work package in the workbasket.

- If the work package ID is specified and *ulWorkOrder* is NULL, the specified work package is retrieved.
- If the work package ID is specified and *ulWorkOrder* is set to SIMWM_ORDER_SYSTEM_NEXT, SIMWM_ORDER_FIFO_NEXT, SIMWM_ORDER_LIFO_NEXT, or SIMWM_ORDER_PRIORITY_NEXT, the next available work package after the one specified is retrieved.
- Once the specified or next work package in the workbasket is retrieved, the work package is not accessible to other users.

Follow-Up Tasks:

- Call **SimWmReturnWorkPackage** to return the work package to the workbasket. This makes the work package available to other users.
- Call **SimWmRouteWorkPackage** to route the work package to another workbasket. This makes the work package available to other users at the destination workbasket.
- When your application no longer needs the SNAPSHOTSTRUCT data, use **SimLibFree** to free the buffer.

Related Functions

- **SimWmReturnWorkPackage**
- **SimWmRouteWorkPackage**

SimWmGetWorkPackagePriority (Get the Priority of a Work Package)

```

Format
SimWmGetWorkPackagePriority( hSession, ulWorkPackageID, ulInstanceID,
pAsyncCtl, pRC )
    
```

Purpose

Use the **SimWmGetWorkPackagePriority** function to determine the priority assigned to a work package. The priority identifies the work order of items located in the workbasket. You can determine the current priority of an item even if the item is locked.

Parameters

- hSession* HSESSION — input
 The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.
- ulWorkPackageID* ULONG — input
 Identifier of the work package that represents the work being done, such as the document being routed.
- ulInstanceID* ULONG — input
 Identifier of the work package instance that distinguishes one parallel path from another within the process.
- pAsyncCtl* PASYNCCTLSTRUCT — input
 Not supported.

SimWmGetWorkPackagePriority

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 1 to indicate that *ulParam1* contains a pointer.

ulParam1 Contains a pointer to a TIMESTAMP buffer that provides the date and time the work package entered the workbasket.

ulParam2 Contains the current priority of the specified work package.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_PRC

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the TIMESTAMP data, use **SimLibFree** to free the buffer.

Related Functions

- **SimWmGetWorkPackage**
- **SimWmSetWorkPackagePriority**
- **SimWmRouteWorkPackage**

SimWmListHistory (List the History of a Work Package)

Format

SimWmListHistory(*hSession*, *ulWorkPackageID*, *ulInstanceID*, *fHistoryRequest*, *pAsyncCtl*, *pRC*)

Purpose

Use the **SimWmListHistory** function to obtain the log of activity for the specified work package.

Parameters

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibSimLibLogon** function creates the session information.

ulWorkPackageID ULONG — input

Identifier of the work package that represents the work being done, such as the document being routed.

ulInstanceID ULONG — input

Identifier of the work package instance that distinguishes one parallel path from another within the process.

<i>fHistoryReques</i>	BITS — input Not supported.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a pointer to an array of WMHISTLOGENTRYSTRUCT structures containing the variable identifiers and values for a specific work package.
<i>ulParam2</i>	Contains the number of variables in the array that <i>ulParam1</i> points to.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_PRC • SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Effects: On successful completion of the function, all history events associated with the work package are returned.

Follow-Up Tasks: When your application no longer needs the work management history information for the specified work package, use the `SimLibFree(hSession, (PVOID)ulParam1, pRC)` function to free the buffer.

Related Functions

- `SimLibLogon`

SimWmListProcesses (List the Processes)

<p>Format</p> <p><code>SimWmListProcesses(hSession, pAsyncCtl, pRC)</code></p>

Purpose

Use the `SimWmListProcesses` function to obtain a list of all existing processes in the Content Manager for iSeries system.

Parameters

<i>hSession</i>	HSESSION — input
-----------------	------------------

SimWmListProcesses

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

pAsyncCtl PASYNCCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 1 to indicate that *ulParam1* contains a pointer.

ulParam1 Contains a pointer to an ITEMNAMESTRUCT array.

ulParam2 Contains the number of elements in the array that *ulParam1* points to.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_ERROR_READING_FROM_FILE
- SIM_RC_FILE_NOT_FOUND
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_PRC
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Exceptions: This function provides all processes defined in the system. Use the **SimWmGetProcessInfo** function with one of the processes that **SimWmListProcesses** returns.

Follow-Up Tasks: When your application no longer needs the process list, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

SimWmListWorkBaskets (List the Workbaskets)

Format

```
SimWmListWorkBaskets( hSession, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmListWorkBaskets** function to get a list of all workbaskets defined in the system.

Parameters

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

pAsyncCtl PASYNCCTLSTRUCT — input

Not supported.

pRC

PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the PRCSTRUCT structure, see “PRCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an PRCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a pointer to an ITEMNAMESTRUCT array.
<i>ulParam2</i>	Contains the number of elements in the array that <i>ulParam1</i> points to.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_ITEM_OR_FOLDER_VALUE • SIM_RC_INVALID_PRC • SIM_RC_LIB_CLIENT_ERROR • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Exceptions: This function does not provide detailed information about the definition of a workbasket. To get that information, use **SimWmGetWorkBasketInfo** with one of the identifiers that **SimWmListWorkBaskets** returns.

Follow-Up Tasks: When your application no longer needs the ITEMNAMESTRUCT array, use **SimLibFree** to free the buffer.

Related Functions

- **SimWmGetWorkBasketInfo**

SimWmMatchEvent (Satisfy an Event for a Work Package)

Format

```
SimWmMatchEvent( hSession, ulActivate, pszProcessID, pszCollectionPointName,  
ulWorkPackageID, ulInstanceID, ulEventType, pszEventCriteria, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmMatchEvent** function to satisfy an event for a work package that is at a collection point.

Parameters

hSession

HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

SimWmMatchEvent

<i>ulActivate</i>	ULONG — input Indicator of whether the collection point should be activated. The valid values are: SIMWWM_ACTIVATE_COLLECTION_POINT Activate the collection point if the work package is not currently at the collection point. SIMWWM_NO_ACTIVATE_COLLECTION_POINT Do not activate the collection point if the work package is not currently at the collection point.
<i>pszProcessID</i>	PSZ — input Pointer to the process identifier.
<i>pszCollectionPointName</i>	PSZ — input Pointer to the name of the collection point.
<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>ulEventType</i>	ULONG — input The type of event to be satisfied at the collection point. The valid values are: SIMWWM_EVENT_INDEX_CLASS The event is the arrival of an item of a specified index class. SIMWWM_EVENT_TIME The event is the expiration of a time period. SIMWWM_EVENT_USERDEF_MIN - SIMWWM_EVENT_USERDEF_MAX The event is a user-defined event.
<i>pszEventCriteria</i>	PSZ — input Pointer to match criteria. If <i>ulEventType</i> is SIMWWM_EVENT_INDEX_CLASS , the match criteria must be an index class identifier. If <i>ulEventType</i> is SIMWWM_EVENT_TIME , this field is ignored and the current system date of the server is used as the match criteria.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “ RCSTRUCT (Return Code Information Structure) ” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • OIM_INVALID_RELEASE_CRITERIA • OIM_INVALID_WF_ITEM • OIM_ITEM_NOT_IN_WORKFLOW • OIM_ITEM_NOT_SUSPENDED • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_USCLASSID_VALUE

Guidelines for Use

This function either satisfies events or activates a work package at a collection point. If an event is matched for a specified work package, that work package event is satisfied. If an event is not matched and the activate flag is set to SIMWM_ACTIVATE_COLLECTION_POINT, the work package is activated at the collection point.

If the last event in an event list of the collection point is satisfied, the work package is released from the collection point and is sent to begin the route specified for that event list in the collection point definition.

Calling this function with event type of SIMWM_EVENT_TIME, causes all collection points to be tested for the date expiration criteria to have been satisfied. This function is equivalent to the Release pending work items function.

SimWmQueryVariables (Query Variables for a Specific Work Package)

Format

```
SimWmQueryVariables( hSession, ulWorkPackageID, ulInstanceID, pAsyncCtl,
pRC )
```

Purpose

Use the **SimWmQueryVariables** function to return all variables and values associated with a specific work package.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
-----------------	---

SimWmQueryVariables

<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package.
<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer. Otherwise, this field contains the value 0.
<i>ulParam1</i>	Pointer to an array of WMVARSTRUCT structures containing the variable identifiers and values for a specific work package.
<i>ulParam2</i>	Contains the number of variables in the array that <i>ulParam1</i> points to.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_PRC

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the work package variable information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

SimWmQueryWorkPackage (Query a Work Package)

Format

```
SimWmQueryWorkPackage( hSession, ulWorkPackageID, ulInstanceID,  
pAsyncCtl, pRC )
```

Purpose

Use the **SimWmQueryWorkPackage** function to retrieve the contents and attributes of a work package.

Parameters

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

<i>ulWorkPackageID</i>	ULONG — input	Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input	Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input	Not supported.
<i>pRC</i>	PRCSTRUCT — input/output	The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer to a data area.
<i>ulParam1</i>	Contains a pointer to a SNAPSHOTSTRUCT data structure that provides the returned item and associated workflow information.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_INDEX_CLASS • SIM_RC_INVALID_PRC • SIM_RC_LIB_CLIENT_ERROR • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the SNAPSHOTSTRUCT data, use **SimLibFree** to free the buffer.

Related Functions

- **SimWmRouteWorkPackage**

SimWmReturnWorkPackage (Return a Work Package to a Workbasket)

Format

```
SimWmReturnWorkPackage( hSession, ulWorkPackageID, ulInstanceID,
usWorkPriority, pAsyncCtl, pRC )
```

SimWmReturnWorkPackage

Purpose

Use the **SimWmReturnWorkPackage** function to return a work package instance that is currently open in a workbasket back to that workbasket. This is the opposite of **SimWmGetWorkPackage**. After using this function, the work package instance is again available.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>usWorkPriority</i>	USHORT — input Priority of the work to perform. The priority affects the work sequencing as the work package moves through a process. A larger number is a higher priority. Use zero to keep the current priority.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_PRC• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects: The application can use this function when the user is unable to complete the work and needs to resume later. **SimWmGetWorkPackage** opens the work package, and **SimWmReturnWorkPackage** closes the work package, making it again available in the workbasket.

Related Functions

- SimWmGetWorkPackage
- SimWmRouteWorkPackage

SimWmRouteWorkPackage (Route a Work Package)**Format**

```
SimWmRouteWorkPackage( hSession, pszWorkBasketID, ulWorkPackageID,  
ulInstanceID, usWorkPriority, fRoute, pszOverrideAction, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmRouteWorkPackage** function to assign a work package to a workbasket, reassign a work package from one workbasket to another, or continue a work package to the next step in a predefined process.

Parameters

<i>hSession</i>	HSESSION — input
	The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pszWorkBasketID</i>	PSZ — input
	Pointer to the name of the workbasket. If NULL and the work package is on a process, the work package will be continued to the next step in the process.
<i>ulWorkPackageID</i>	ULONG — input
	Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input
	Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>usWorkPriority</i>	USHORT — input
	Priority of the work to be performed. The priority affects the work sequencing of the work package at the workbasket. A larger number is a higher priority. Use a priority of zero to request the default priority.
<i>fRoute</i>	BITS — input
	Work package routing control. Valid value is: SIMWM_IGNORE_OVERLOAD If NULL, workbasket overload limits will be checked.
<i>pszOverrideAction</i>	PSZ — input
	Pointer to the name of the action list to use when work package is routed to the next workbasket. This action list overrides the default action list associated with the next workbasket.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input

SimWmRouteWorkPackage

Not supported.

pRC

PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Always zero.
<i>ulParam1</i>	Contains the work package ID.
<i>ulParam2</i>	Contains the work package instance.
<i>ulRC</i>	Contains one of the following: <ul style="list-style-type: none">• SIM_RC_OK• OIM_INVALID_FOVERLOAD_VALUE• OIM_WB_FULL• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

This function can be used to continue an item on a process, assign an item to a workbasket, or reassign an item to another workbasket. If the SIMWM_IGNORE_OVERLOAD is not set and *pszWorkBasketID* is NULL, the item will be added to the workbasket even when an overload condition exists; however, the application will be notified of the overload condition. This function can be used in combination with **SimWmQueryWorkPackage** to determine the location of the work package before routing the work package.

Exceptions: If a work package is at a collection point, it cannot be routed until the events for the collection point are satisfied.

Related Functions

- **SimWmCreateWorkPackage**
- **SimWmQueryWorkPackage**

SimWmSetWorkPackagePriority (Set the Priority of a Work Package)

Format

```
SimWmSetWorkPackagePriority( hSession, ulWorkPackageID, ulInstanceID,  
usPriority, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmSetWorkPackagePriority** function to set the priority of a work package. This priority can control the work order of work packages in the workbasket.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>usPriority</i>	USHORT — input Priority of the work to be performed. The priority affects the work sequencing of the work package. A larger number is a higher priority. Use a priority of zero to request the default priority.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_PRC • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Exceptions: The priority value can be between 1 and 65,535. The Content Manager for iSeries Client Application, however, only supports values between 1 and 31,999.

Related Functions

- **SimWmGetWorkPackage**
- **SimWmGetWorkPackagePriority**
- **SimWmRouteWorkPackage**

SimWmSuspendWorkPackage (Suspend a Work Package)

Format

```
SimWmSuspendWorkPackage( hSession, ulWorkPackageID, ulInstanceID,
pSuspendCriteria, pAsyncCtl, pRC )
```

Purpose

Use the **SimWmSuspendWorkPackage** function to suspend a work package instance that is currently in a workbasket, and cause the work package to remain unselectable until its suspend criteria are satisfied or the work package is explicitly reactivated.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>ulWorkPackageID</i>	ULONG — input Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — input Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>pSuspendCriteria</i>	PWMSUSPENDSTRUCT — input Pointer to a single WMSUSPENDSTRUCT structure containing the criteria for suspension and release of a work package.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following: <ul style="list-style-type: none"> • SIM_RC_OK • OIM_INVALID_READY_WB • OIM_INVALID_RELEASE_CRITERIA • SIM_RC_COMPLETION_ERROR

- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_ITEM_ID
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Preparation:

- You can specify up to 8 index classes in the suspension criteria.
- You can suspend folders pending the arrival of other items of a specified index class, or until a period of time has expired.
- If you suspend for a specified index class(es), you must also specify a period of time.
- If you specify SIM_INDEX_ANY as the index class in the release criteria, the item will be suspended for the arrival of an item belonging to any index class defined in the system.

Effects:

- When the release criteria are satisfied, a formerly suspended item is assigned to the workbasket associated with those criteria in the WMSUSPENDSTRUCT data structure.

Exceptions:

- The item to suspend must be in a workbasket.
- SIMWM_NEXT is not a valid workbasket when an item is on an ad hoc process.
- Changes to the suspension state of an item do not change the checkout or access status of the item. If your application checks out an item and suspends it, it is the responsibility of the application to be sure that the item is checked in. When the item meets the release criteria, it becomes active and, if your application did not check the item in, it remains checked out by your application.
- If SIM_INDEX_ANY is entered as an index class, no other index class can be defined in the suspend criteria.
- If the item is currently suspended and **SimWmSuspendWorkPackage** is issued, the item will not be suspended again. The new suspend request will be ignored and the application will receive a successful completion.

Sim400ConvertCodepage (Code Page Conversion)

Format

```
Sim400ConvertCodepage( hSession, iConvertDirection, chInputBuffer,  
chOutputBuffer, ulInputSize, ulOutputSize, pAsyncCtl, pRC )
```

Purpose

Use the **Sim400ConvertCodepage** function to handle code page conversion between the workstation and the iSeries.

Parameters

hSession HSESSION — input

The handle to the Content Manager for iSeries session information. The **SimLibLogon** function creates the session information.

iConvertDirection

INT — input

Specify one of the following:

SIM_400_CONVERT_TO400

SIM_400_CONVERT_FROM400

chInputBuffer

CHAR — input

The buffer to send to the server.

chOutputBuffer

CHAR — input

Space for returned data.

ulInputSize

ULONG — input

Length of the buffer that is being sent to the server. Maximum size is 32,700.

ulOutputSize

ULONG — input

Size of the space for returned data.

pAsyncCtl

PASYNCTLSTRUCT — input

Not supported.

pRC

PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

usParam

The function does not use this field.

ulParam1

Contains the length of the output buffer.

ulParam2

The function does not use this field.

ulRC

Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_LIB_CLIENT_ERROR

Related Functions

- Sim400SendReceive

Sim400SendReceive (Send Data to AS/400)

Format

```
Sim400SendReceive( hSession, chInputBuffer, chOutputBuffer, ulInputSize, ulOutputSize, pAsyncCtl, pRC )
```

Purpose

Use the **Sim400SendReceive** function to send up to 32,700 bytes of data to the iSeries. The data sent to the server can be processed by a customer-written application, and the results can be returned to the workstation.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>chInputBuffer</i>	CHAR — input The buffer to send to the server.
<i>chOutputBuffer</i>	CHAR — input Space for returned data.
<i>ulInputSize</i>	ULONG — input Length of the buffer that is being sent to the server. Maximum size is 32,700.
<i>ulOutputSize</i>	ULONG — input Size of the space for returned data.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	Contains the number of bytes received.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR

Example

Refer to sample program QVIRCVSND in source file QLBLSRC in your QVI library. This sample program shows a COBOL program that receives data from the **Sim400SendReceive** function, and returns data to the function.

Related Functions

- **Sim400ConvertCodepage**

Ip2CloseTOC (Close a Table of Contents)

Format

```
Ip2CloseTOC( hSession, hTOC, pAsyncCtl, pRC )
```

Purpose

Use the **Ip2CloseTOC** function to close the specified table of contents and then release the table-of-contents handle.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hTOC</i>	HTOC — input The handle to the table of contents you want to close. Use the SimLibGetTOC function to get this handle.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	The function does not use this field.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_LIB_CLIENT_ERROR• SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Effects:

- After you use this function to close the table of contents, you cannot use the table-of-contents handle (*hTOC*) again.
- Use the **SimLibGetTOC** function to get a new table-of-contents handle.

Related Functions

- Ip2CloseToc
- Ip2GetTOCUpdates
- Ip2TOCCount
- Ip2TOCStatus
- SimLibGetItemAffiliatedTOC
- SimLibGetTOC

Ip2GetLibSessionInfo (Get the Information for a Library Session)

Format

Ip2GetLibSessionInfo(*hSession*, *pAsyncCtl*, *pRC*)

Purpose

Use the **Ip2GetLibSessionInfo** function to return information for the current library session.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pAsyncCtl</i>	PASYNCCTLSTRUT — input Not supported.
<i>pRC</i>	PRSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a pointer to a buffer with a LIBSESSIONINFOSTRUCT data structure. For more information, on this data structure, see “LIBSESSIONINFOSTRUCT (Library Session Information Structure)” on page 148.
<i>ulParam2</i>	This function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_PRC

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the LIBSESSIONINFOSTRUCT data, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to free the buffer.

Ip2GetTOCUpdates (Get the Updates to a Table of Contents)**Format**

```
Ip2GetTOCUpdates( hSession, hTOC, usUpdate, pAsyncCtl, pRC )
```

Purpose

Use the **Ip2GetTOCUpdates** function to refresh a table of contents that you received from a previous **SimLibGetTOC** function.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
-----------------	---

Ip2GetTOCUpdates

<i>hTOC</i>	HTOC — input The handle to the table of contents that you want to refresh. Use the SimLibGetTOC function to get this handle.
<i>usUpdate</i>	USHORT — input Not supported.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the total number of items in the table of contents.
<i>ulParam1</i>	Contains a pointer to a buffer with an array of TOCENTRYSTRUCT data structures which indicates the number of items that have been updated, deleted, or added. For more information on the TOCENTRYSTRUCT data structure, see “TOCENTRYSTRUCT (Table of Contents Entry Data Structure)” on page 157.
<i>ulParam2</i>	Contains the handle to the table of contents.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• OIM_INVALID_FUPDATE_VALUE• OIM_INVALID_HTOC_VALUE• SIM_RC_COMMUNICATIONS_ERROR• SIM_RC_COMPLETION_ERROR• SIM_RC_INVALID_HSESSION• SIM_RC_INVALID_ITEM_ID• SIM_RC_INVALID_POINTER• SIM_RC_INVALID_PRC• SIM_RC_LIB_CLIENT_ERROR• SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Follow-Up Tasks: When your application no longer needs the table of contents, use the **Ip2CloseTOC** function to close the table of contents and release the handle.

Related Functions

- **SimLibGetTOC**
- **Ip2CloseTOC**
- **Ip2TOCStatus**
- **Ip2GetTOCUpdates**

Ip2ListAttrs (List the User-Defined Attributes)

Format

```
Ip2ListAttrs( hSession, pAsyncCtl, pRC )
```

Purpose

Use the **Ip2ListAttrs** function to get a list of the attributes in the system.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	If the <i>ulParam2</i> field contains a value greater than 0, this field contains a pointer to a buffer with a NAMESTRUCT array. Each element in this array provides the index attribute identifiers that are associated with a specific attribute name. For more information on this data structure, see “NAMESTRUCT (Name Data Structure)” on page 149.
<i>ulParam2</i>	Contains the number of elements in the array that <i>ulParam1</i> points to.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_LIB_CLIENT_ERROR • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects:

- Use the **SimLibGetAttrInfo** function to get additional information about a specific index attribute.
- Attributes with negative IDs or those greater than 32767 are system attributes. You cannot modify these.

Ip2ListAttrs

- If an attribute has not been defined to any index class, it is not returned by **Ip2ListAttrs**.

Follow-Up Tasks: When your application no longer needs the array of index attribute identifiers, use the **SimLibFree(hSession, (PVOID)ulParam1, pRC)** function to free the buffer.

Related Functions

- **SimLibGetAttrInfo**

Ip2ListContentClasses (List the Content Classes)

Format

```
Ip2ListContentClasses( hSession, usContentClassType, pAsyncCtl, pRC )
```

Purpose

Use the **Ip2ListContentClasses** function to display the content class records that are in the library server database.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>usContentClassType</i>	USHORT — input The type of content classes to list. The valid values are: OIM_SA_ALL_CC Lists both the IBM-defined content classes and the user-defined content classes. OIM_SA_IBM_CC Lists only the IBM-defined content classes. OIM_SA_USR_CC Lists only the user-defined content classes.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in the RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1, to indicate that <i>ulParam1</i> contains a pointer. If no records exist for the specified content class type, this field contains the value 0.
<i>ulParam1</i>	Contains a pointer to the array of CONTENTCLASSINFO data structures containing the list of content classes. For more

information on this data structure, see “CONTENTCLASSINFO (Content Class Information Structure)” on page 142. If no records exist for the specified content class type, this field contains the value NULL.

<i>ulParam2</i>	Contains the number of content classes in the library server database. If <i>ulRC</i> contains an error code, <i>ulParam2</i> contains the value NULL.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_CC_TYPE • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR • SIM_RC_QUERY_FAILED

Guidelines for Use

Follow-Up Tasks: When you finish with the content class information, use the **SimLibFree**(*hSession*, (PVOID)*ulParam1*, *pRC*) function to release allocated storage.

Ip2ListServers (List the Accessible Servers)

Format

```
Ip2ListServers( pServerInfo, ulServerInfoSize, fSrchfilter, pRC )
```

Purpose

Use the **Ip2ListServers** function to retrieve information about all the servers accessible to the system. You can use this function to determine the eligible libraries to display as part of a logon interaction.

Parameters

<i>pServerInfo</i>	PSERVERINFOSTRUCT — input/output The pointer to a buffer that contains an array of server names and types. The calling application allocates memory for this structure.
<i>ulServerInfoSize</i>	ULONG — input The size, in bytes, of the buffer allocated for the SERVERINFOSTRUCT array.
<i>fSrchfilter</i>	ULONG — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1.
<i>ulParam1</i>	If <i>usParam</i> contains a value greater than 0, this field contains a pointer to an array of SERVERINFOSTRUCT data structures. "Guidelines for Use" explains how the value of the <i>ulServerInfoSize</i> parameter affects the value returned in <i>ulParam1</i> . For more information on the SERVERINFOSTRUCT data structure, see "SERVERINFOSTRUCT (Server Information Structure)" on page 153.
<i>ulParam2</i>	Contains the number of the servers returned by this call, though not necessarily the number of servers in the system.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none">• SIM_RC_OK• OIM_INVALID_PSERVERINFO_PTR• OIM_RC_INPUTBUF_TOO_SMALL• OIM_RC_ISO_CONNECT_FAILED• OIM_RC_ISO_LISTSVR_FAILED

Guidelines for Use

Exceptions:

- Your application can connect to all the servers, but not necessarily log on to all of them. You must have a valid user ID and password to access the database on the server.
- If the input value of *ulServerInfoSize* is too small to receive the data, error code OIM_RC_INPUTBUF_TOO_SMALL is returned, and the *ulParam2* field of the RCSTRUCT data structure contains the number of servers found.

Related Functions

None

Ip2QueryClassPriv (Query the Privilege String for an Index Class or View)

Format

```
Ip2QueryClassPriv( hSession, usClassType, usID, pAsyncCtl, pRC )
```

Purpose

Use the **Ip2QueryClassPriv** function to return the evaluated privilege string for the index class that you specify. The evaluated privilege string indicates your access rights to the information in the system. You should use it with **Ip2QueryPrivBuffer** to determine access rights.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>usClassType</i>	USHORT — input

	Not supported.
<i>usID</i>	USHORT — input The ID of an index class.
<i>pAsyncCtl</i>	PASYNCCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in the RCSTRUCT data structure:

<i>usParam</i>	This parameter contains the value 1 to indicate that <i>ulParam1</i> contains a pointer.
<i>ulParam1</i>	Contains a PSZ pointer. This pointer identifies the location of a CHAR <i>szPrivilege</i> [401] buffer where a data structure contains the evaluated privilege string.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_CLASS_TYPE • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_INVALID_USCLASSID_VALUE • SIM_RC_LIB_CLIENT_ERROR • SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Effects:

- The privilege string is evaluated for the class with respect to the user who got the *hSession* by logging on. The evaluated privilege string specifies the privileges of that user for the specified index class as computed by the access control algorithm.

Follow-Up Tasks: When your application no longer needs the data structure that *ulParam1* points to, use the **SimLibFree**(*hSession*,(PVOID)*ulParam1*, *pRC*) function to free the data structure.

Ip2QueryPrivBuffer (Query a Privilege Buffer)

Format

```
Ip2QueryPrivBuffer( pszPrivilege, ulAuthority, pRC )
```

Purpose

Use the `Ip2QueryPrivBuffer` function to determine whether a certain authority is granted in a specified privilege buffer.

Parameters

<i>pszPrivilege</i>	PSZ — input
	The current privileges set for the user.
<i>ulAuthority</i>	ULONG — input
	The general privilege to search for. The valid values are:
	OIM_ACL Determines the authority to create, update, and delete access lists.
	OIM_ADD_ITEMS_TO_WB Determines the authority to add an item to a workbasket.
	OIM_ADD_ITEMS_TO_WF Determines the authority to add an item to a workflow.
	OIM_ADD_NEW_BASE_PART Determines the authority to add a new document.
	OIM_ADD_NOTE_TO_NOTELOG Determines the authority to add a note object to the note log.
	OIM_ATTRS Determines the authority to create, update, and delete attributes.
	OIM_CC Determines the authority to create, update, list and delete content classes.
	OIM_CHANGE_INDEX_CLASS Determines the authority to change the index class of any items.
	OIM_CHANGE_ITEMS_TO_WB Determines the authority to change the priority of an item in a workbasket.
	OIM_CHANGE_ITEMS_TO_WF Determines the authority to change an item from the current workflow to a new workflow.
	OIM_CHECK_IN_OUT_ITEMS Determines the authority to check in and check out a folder or document.
	OIM_CLASS Determines the authority to add and delete indexes on an index classes and query their DLLs.
	OIM_CREATE_ITEMS Determines the authority to create a folder or document.
	OIM_DB_UTILITY Determines the authority to allow UTILITY to access the database.

OIM_DELETE_BASE_PART

Determines the authority to delete a document.

OIM_DELETE_ITEMS

Determines the authority to delete a folder or document.

OIM_EXPORT

Determines the authority to export and to send mail that includes an object.

OIM_FAXIN

Determines the authority to receive a facsimile.

OIM_FAXOUT

Determines the authority to send a facsimile.

OIM_FAXSERVER

Determines the authority of the fax server to send or receive a facsimile.

OIM_FILEROOM

Determines the authority to access an application-defined fileroom.

OIM_IMPORT

Determines the authority to import and to receive mail.

OIM_LBOS_BACKUP

Determines the authority to back up the LAN-based object server.

OIM_LIB_SERV_BACKUP

Determines the authority to back up the library server.

OIM_LIB_SERV_CONFIG

Determines the authority to control the library server configuration.

OIM_LICENSE

Determines the authority to update the license information in the database.

OIM_LINK_ITEMS

Determines the authority to add a link between items and a folder.

OIM_OCR

Determines the authority to use an optical character recognition device.

OIM_PRINT

Determines the authority to print.

OIM_PRIV_SET

Determines the authority to create, update, and delete privilege sets.

OIM_READ_BASE_PART

Determines the authority to read a document part.

OIM_READ_HISTORY

Determines the authority to read a history event.

OIM_READ_NOTELOG

Determines the authority to read the note log.

OIM_READ_TOC

Determines the authority to read the folder table of contents.

OIM_READ_WORKBASKET

Determines the authority to get the workbasket information.

OIM_REMOVE_ITEMS_TO_WB

Determines the authority to remove an item from a workbasket.

OIM_REMOVE_ITEMS_TO_WF

Determines the authority to remove an item from a workflow.

OIM_REMOVE_LINKS

Determines the authority to delete a link between items and a folder.

OIM_SA_NLS

Determines the authority to update the supported languages in the database.

OIM_SA_OBJSERV

Determines the authority to update the object server information in the database.

OIM_SA_USER

Determines the general logon privileges of a user.

OIM_SA_WORKBASKET

Determines the authority to create, update, and delete workbaskets.

OIM_SA_WORKFLOW

Determines the authority to create, update, and delete workflows.

OIM_SCAN

Determines the authority to scan images.

OIM_SEARCH_INDEX_INFO

Determines the authority to read user-defined attributes for all index classes and all items in each index class.

OIM_SERVER

Determines the authority to act as a client on behalf of other clients.

OIM_SMS

Determines the authority to manage system-managed storage for a LAN-based object server.

OIM_SNAPSHOT_ALL

Determines the authority to use the **SimLibGetItemSnapshot** or **SimLibGetTOCData** functions on items.

OIM_SUPER_ADMIN

Determines the authority to bypass the access list.

OIM_SUSP_AND_ACTIVATE_ITEMS

Determines the authority to suspend and activate a folder or document.

OIM_UPDATE_AVT_INFO

Determines the authority to update user-defined attribute values for all index classes and all items in each index class.

OIM_UPDATE_BASE_PART

Determines the authority to update a document.

OIM_UPDATE_NOTELOG

Determines the authority to update or delete notes in the note log.

OIM_USER_GROUPS

Determines the authority to create, update, and delete user groups.

OIM_USER_ID

Determines the authority to create, update, and delete user IDs.

OIM_VIEW

Determines the authority to create, update, and delete views.

OIM_WORKFLOW_CONTINUE

Determines the authority to continue an item to the next step of a process.

OIM_WORKFLOW_FORCE_CONTINUE

Determines the authority to force an item, with outstanding events pending, to the next step of a process.

OIM_WORKFLOW_SEARCH

Determines the authority to search a process for items.

pRC

PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in the RCSTRUCT data structure:

<i>usParam</i>	Contains the value 1 if the privilege set represented by <i>pszPrivilege</i> contains the specified authority. Otherwise the field contains the value 0.
<i>ulParam1</i>	The function does not use this field.
<i>ulParam2</i>	The function does not use this field.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • OIM_INVALID_PSZPRIVILEGE_STRING • SIM_INVALID_ULAUTHORITY

Ip2TOCCount (Count the Items in a Table of Contents)

Format

```
Ip2TOCCount( hSession, pItemidItem, usItemType, usWipFilter, usSuspendFilter,
usNbrOfClasses, pusClassIdList, pAsyncCtl, pRC )
```

Purpose

Use the **Ip2TOCCount** function to get a count of the items in a folder or workbasket that satisfy the filtering criteria that you specify. This function is similar to **SimLibGetTOC**, except that this function returns only a count of the items rather than a table of contents. The count includes all items, regardless of authority.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>pItemidItem</i>	PITEMID — input The pointer to an item ID of a folder or workbasket.
<i>usItemType</i>	USHORT — input The type of items to count. Here are the valid values: SIM_DOCUMENT Counts documents. SIM_FOLDER Counts folders. SIM_ALL Counts all types of items.
<i>usWipFilter</i>	USHORT — input Not supported.
<i>usSuspendFilter</i>	USHORT — input Not supported.
<i>usNbrOfClasses</i>	USHORT — input The number of index class identifiers in the list you specify as the value of the <i>pusClassIdList</i> parameter. Specify the value 0 for the <i>usNbrOfClasses</i> parameter to indicate that class is not a criterion for selecting items to count.
<i>pusClassIdList</i>	PUSHORT — input The pointer to a list of index class identifiers that indicate the items to count. You can specify the value NULL for this parameter if you also specify the value 0 for the <i>usNbrOfClasses</i> parameter.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input Not supported.
<i>pRC</i>	PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

<i>usParam</i>	Contains the value 0.
<i>ulParam1</i>	Contains the count of items in the table of contents. If no items satisfy the filtering criteria, this field contains the value 0.
<i>ulParam2</i>	Contains the value 0.
<i>ulRC</i>	Contains one of the following return codes: <ul style="list-style-type: none"> • SIM_RC_OK • SIM_RC_COMMUNICATIONS_ERROR • SIM_RC_COMPLETION_ERROR • SIM_RC_INVALID_HSESSION • SIM_RC_INVALID_POINTER • SIM_RC_INVALID_PRC • SIM_RC_LIB_CLIENT_ERROR • SIM_RC_OUT_OF_MEMORY • SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects: If the item is not a folder or a workbasket, the function returns SIM_RC_INVALID_ITEM_TYPE.

Related Functions

- Ip2GetTOCUpdates
- SimLibGetTOC

Ip2TOCStatus (Get the Status of a Table of Contents)

Format

```
Ip2TOCStatus( hSession, hTOC, usCheck, pAsyncCtl, pRC )
```

Purpose

Use the **Ip2TOCStatus** function to return a value that indicates whether or not a table of contents has been changed.

Parameters

<i>hSession</i>	HSESSION — input The handle to the Content Manager for iSeries session information. The SimLibLogon function creates the session information.
<i>hTOC</i>	HTOC — input The handle to the table of contents for which you want to check the status. The SimLibGetTOC function returns this handle.
<i>usCheck</i>	USHORT — input Not supported.
<i>pAsyncCtl</i>	PASYNCTLSTRUCT — input

Not supported.

pRC PRCSTRUCT — input/output

The pointer to the return data structure. For more information on the RCSTRUCT structure, see “RCSTRUCT (Return Code Information Structure)” on page 151.

Return Values

On successful completion, this function returns values to the following fields in an RCSTRUCT data structure:

usParam Contains the value 0.

ulParam1 If the table of contents has changed, this field contains the value TRUE. If there are no changes, this field contains the value FALSE.

ulParam2 Contains the value 0.

ulRC Contains one of the following return codes:

- SIM_RC_OK
- OIM_EMPTY_WORKBASKET
- OIM_INVALID_HTOC_VALUE
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_HSESSION
- SIM_RC_INVALID_ITEM_ID
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- SIM_RC_LIB_CLIENT_ERROR
- SIM_RC_OUT_OF_MEMORY

Guidelines for Use

Exceptions: This function tells whether a table of contents has changed, but it does not return the updates. After you use the function, your application can use other functions to get the changes themselves. Because the time required for this function is nearly the same as the time required for **SimLibGetTOC** or **SimLibGetTOCUpdates**, you should use those functions instead, if possible.

- Use the **Ip2GetTOCUpdates** function to refresh the table of contents.
- Use the **Ip2CloseTOC** function to close the open table of contents and then use the **SimLibGetTOC** function to refresh the table of contents to reflect the values in the database.

Related Functions

- **Ip2CloseTOC**
- **Ip2GetTOCUpdates**
- **SimLibGetTOC**

Chapter 4. Common Data Structures

This part provides more detailed reference information that describes the common data structures and database tables used for Content Manager for iSeries. The data structures are listed alphabetically and are always in UPPERCASE in the Content Manager for iSeries code. The following information is provided about each data structure:

- Purpose
- Valid fields
- Valid field values
- Usage guidelines

Data Structures

AFFTOCENTRYSTRUCT (Affiliated Table of Contents Entry Structure)

This data structure provides information about which objects are affiliated with an item. It consists of the following:

```
typedef struct _AFFTOCENTRYSTRUCT
{
    ULONG                ulStruct;
    ANNOTATIONSTRUCT    AnnotationData;
    ULONG                ulObjType;
    OBJ                  Obj;
    ULONG                ulObjConCls;
    ULONG                ulObjLength;
    LONG                 lObjSeqAfter;
    ULONG                ulObjFlags;
    TIMESTAMP            tsCreate;
    TIMESTAMP            tsChanged;
} AFFTOCENTRYSHOTSTRUCT, *PAFFTOCENTRYSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>AnnotationData</i>	ANNOTATIONSTRUCT — output The information associated with an annotation object. For more information, see ANNOTATIONSTRUCT (Annotation Information Structure).
<i>ulObjType</i>	ULONG — output The type of object. The valid values are:

AFFTOCENTRYSTRUCT

SIM_ANNOTATION

Indicates that the item is an annotation associated with a folder or a document.

SIM_BASE

Indicates that the object is a base object such as a Mixed Object Document Content Architecture (MO:DCA) or Tag Image File Format (TIFF) file, and is not an annotation, note, or event associated with a folder or document.

SIM_NOTE

Indicates that the item is a note associated with a folder or a document.

<i>Obj</i>	OBJ — output The object handle data structure that identifies the object. For more information, see HOBJ (Handle to Query Stored Object).
<i>ulObjConCls</i>	ULONG — output The object content class of the object you query. The value SIM_CC_UNKNOWN indicates the undefined content class.
<i>ulObjLength</i>	ULONG — output The length of the object in bytes.
<i>lObjSeqAfter</i>	LONG — output The order of the object relative to other objects in the item. Restriction: This is the value of the unsupported <i>lSeqAfterPart</i> parameter of the SimLibCreateObject function.
<i>ulObjFlags</i>	ULONG — output Not supported.
<i>tsCreate</i>	TIMESTAMP — output The date and time that the item or object was created.
<i>tsChanged</i>	TIMESTAMP — output The date and time that the item or object was changed.

ANNOTATIONSTRUCT (Annotation Information Structure)

This data structure provides information about an annotation affiliated with an object. It consists of the following:

```
typedef struct _ANNOTATIONSTRUCT
{
    ULONG                ulStruct;
    ULONG                ulPart;
```

```

ULONG          ulPageNumber;
USHORT        usX;
USHORT        usY;
USHORT        usT;
USHORT        usAnnotUnused;

```

```
} ANNOTATIONSTRUCT, *PANNOTATIONSTRUCT;
```

Fields

<i>ulStruct</i>	<p>ULONG — input/output</p> <p>The length of the structure in bytes, including the length of this field.</p>
<i>ulPart</i>	<p>ULONG — input/output</p> <p>The part number of the object. Only positive values are valid.</p>
<i>ulPageNumber</i>	<p>ULONG — input/output</p> <p>The page number that the annotation object refers to.</p>
<i>usX</i>	<p>USHORT — input/output</p> <p>The X coordinate for the annotation object on the page that the value of the <i>ulPageNumber</i> field references.</p>
<i>usY</i>	<p>USHORT — input/output</p> <p>The Y coordinate for the annotation object on the page that the value of the <i>ulPageNumber</i> field references.</p>
<i>usT</i>	<p>USHORT — input/output</p> <p>Not supported.</p>
<i>usAnnotUnused</i>	<p>USHORT — input/output</p> <p>A reserved field.</p>

ATTRINFOSTRUCT (Attribute Information Structure)

This structure provides the data needed to create, modify, and list a user-defined attribute. It consists of the following:

```

typedef struct _ATTRINFOSTRUCT
{
    ULONG          ulStruct;
    BOOL          fUseBidirectional;
    BOOL          fSymmetricSwapping;
    BOOL          fShaping;
    LONG          lMin;
    LONG          lMax;
    BITS          fTypeFlags;
    USHORT        usAttrType;
    USHORT        usHorizontalOrientation;

```

ATTRINFOSTRUCT

USHORT	<i>usVerticalOrientation</i> ;
USHORT	<i>usMode</i> ;
USHORT	<i>usNumericSelectionDefault</i> ;
CHAR	<i>szAttributeName</i> ;
CHAR	<i>achLanguageCode</i> ;

} ATTRINFOSTRUCT, *PATTRINFOSTRUCT;

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>fUseBidirectional</i>	BOOL — output This is always set to FALSE.
<i>fSymmetricSwapping</i>	BOOL — input This is always set to FALSE.
<i>fShaping</i>	BOOL — input This is always set to FALSE.
<i>lMin</i>	LONG — input The meaning of <i>lMin</i> varies with the value of the <i>usAttrType</i> parameter: <ul style="list-style-type: none">• It is the minimum length of the string and must contain the value 0 or a greater value, if <i>usAttrType</i> contains SIM_ATTR_FSTRING.• When the data could be a double byte character string (DBCS), space must be allowed for the possible use of the shift in (SI) and the shift out (SO) indicators in a mixed string situation.• It is the minimum value allowed if <i>usAttrType</i> contains SIM_ATTR_LONG.
<i>lMax</i>	LONG — output The meaning of <i>lMax</i> varies with the value of the <i>usAttrType</i> parameter: <ul style="list-style-type: none">• It is the maximum length of the string and must contain a value greater than 0 and greater than <i>lMin</i>, if <i>usAttrType</i> contains SIM_ATTR_FSTRING.• It is the maximum value allowed if <i>usAttrType</i> contains SIM_ATTR_LONG.
<i>fTypeFlags</i>	BITS — output Not supported.
<i>usAttrType</i>	USHORT — output In Content Manager for iSeries, this is always set to SIM_ATTR_VSTRING.
<i>usHorizontalOrientation</i>	USHORT — output

	Not supported.
<i>usVerticalOrientation</i>	USHORT — output
	Not supported.
<i>usMode</i>	USHORT — output
	Not supported.
<i>usNumericSelectionDefault</i>	USHORT — output
	Not supported.
<i>szAttributeName</i>	CHAR[SIM_ATTR_NAME_LENGTH+1] — input/output
	A NULL-terminated character string containing the application-defined name of the attribute.
<i>achLanguageCode</i>	CHAR[SIM_LANGUAGE_CODE_LENGTH+1] — output
	The 3-character national language code for this attribute name. The values for language codes are described in the <i>IBM National Language Design Guide: National Language Support Reference Manual Volume 2</i> .

ATTRLISTSTRUCT (Attribute List Data Structure)

This data structure defines a single system-defined or user-defined attribute value to be associated with an item. The structure is also used when creating an item. It consists of the following:

```
typedef struct _ATTRLISTSTRUCT
{
    ULONG                ulStruct;
    PSZ                  pszAttributeValue;
    BITS                 fAttrFlags;
    USHORT               usAttrId;
    USHORT               usAttrType;
} ATTRLISTSTRUCT, *PATTRLISTSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — input/output
	The length of the structure in bytes, including the length of this field.
<i>pszAttributeValue</i>	PSZ — input/output
	The pointer to a NULL-terminated character string containing the value of an attribute.
<i>fAttrFlags</i>	BITS — output
	Flags that denote attribute characteristics. These flags indicate whether the attribute value is accessible for reading, writing, or both, and whether it is required for the index class. The valid

ATTRLISTSTRUCT

values follow. You can use a bit-wise inclusive OR operator (|) to combine them.

SIM_ATTR_READABLE

Indicates that the attribute is accessible for reading for this index class.

SIM_ATTR_READWRITE

Indicates that the attribute is accessible for both reading and writing for this index class.

SIM_ATTR_WRITEABLE

Indicates that the attribute is accessible for writing for this index class.

SIM_ATTR_ALLOW_NULL

Indicates that the attribute value is not required for this index class.

usAttrId

USHORT — input/output

The unique identifier of an attribute. See the note the follows this list for a discussion of the Content Manager for iSeries system-defined attributes.

usAttrType

USHORT — input/output

In Content Manager for iSeries, this is always set to SIM_ATTR_VSTRING.

Content Manager for iSeries supports the system-defined attributes shown in Table 2.

Table 2. Source of Values for System-Defined Attributes

Attribute Name	Description	How Assigned
OIM_ID_ITEM_CREATE_TIMESTAMP	The timestamp when the item was created	System-assigned and system-maintained automatically
OIM_ID_ITEM_NAME	The name of the item	You can assign when creating an item and update when opening an item for read and write access
OIM_ID_SYS_MOD_TIMESTAMP	The timestamp for changes to the system-assigned or user-defined attributes of the item	System-assigned and system-maintained automatically
OIM_ID_ITEM_ID	The item ID of the item	System-assigned and system-maintained automatically

CLASSATTRSTRUCT (Class Attribute Structure)

This data structure contains specific information about the attributes defined for an index class. It consists of the following:

```
typedef struct _CLASSATTRSTRUCT
{
```

```

ULONG                ulStruct;
BOOL                 fAttrRequiredField;
BITS                 fAttrAccess;
USHORT               usAttrId;

```

```

} CLASSATTRSTRUCT, *PCCLASSATTRSTRUCT;

```

Fields

<i>ulStruct</i>	<p>ULONG — output</p> <p>The length of the structure in bytes, including the length of this field.</p>
<i>fAttrRequiredField</i>	<p>BOOL — output</p> <p>A flag that indicates whether a value is required for this attribute. The valid values are:</p> <p>TRUE Indicates that a value is required.</p> <p>FALSE Indicates that a value is not required.</p> <p>Restriction: This field is valid for index classes only. It is not valid for views.</p>
<i>fAttrAccess</i>	<p>BITS — output</p> <p>A flag that indicates the type of access for the attribute. This field is valid only for views. It is not valid for index classes. The valid values are:</p> <p>SIM_ATTR_READABLE Indicates read access.</p> <p>SIM_ATTR_READWRITE Indicates read and write access. This value is a combination of SIM_ATTR_READABLE and SIM_ATTR_WRITEABLE.</p> <p>SIM_ATTR_WRITEABLE Indicates write access.</p>
<i>usAttrId</i>	<p>USHORT — output</p> <p>The unique identifier of an attribute.</p>

CLASSINDEXATTRSTRUCT (Class Index Attribute Structure)

This data structure contains information about an attribute within an index on an index class attributes table. It consists of the following:

```

typedef struct _CLASSINDEXATTRSTRUCT
{
    ULONG                ulStruct;
    USHORT               usAttrId;
    USHORT               usIndexSortOrder;
} CLASSINDEXATTRSTRUCT, *PCCLASSINDEXATTRSTRUCT;

```

CLASSINDEXATTRSTRUCT

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>usAttrId</i>	USHORT — output The unique identifier of an attribute. The attribute can be user-defined but not system-defined, and it must be in the index class for which this index is requested.
<i>usIndexSortOrder</i>	USHORT — output In Content Manager for iSeries, this is always set to SIM_INDEX_ASCENDING.

CLASSINDEXSTRUCT (Class Index Structure)

This data structure contains the index class attributes that are used to create a database index on an index class. It consists of the following:

```
typedef struct _CLASSINDEXSTRUCT
{
    ULONG                ulStruct;
    BITS                fIndexFlags;
    PCLASSINDEXATTRSTRUCT pClassIndexAttr;
    USHORT              usNbrAttrIds;
    CHAR                szIndexName;
} CLASSINDEXSTRUCT, *PCLASSINDEXSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>fIndexFlags</i>	BITS — output Not supported.
<i>pClassIndexAttr</i>	PCLASSINDEXATTRSTRUCT — output A pointer to a ClassIndexAttrStruct data structure containing class index attribute information. For more information, see CLASSINDEXATTRSTRUCT (Class Index Attribute Structure).
<i>usNbrAttrIds</i>	USHORT — output The number of attribute IDs in the ClassIndexAttrStruct structure.
<i>szIndexName</i>	CHAR[<i>SIM_INDEX_NAME_LENGTH</i> +1] — output The unique name of an index class database index.

CLASSINFOSTRUCT (Index Class Information Structure)

This data structure provides information about an index class. It consists of the following:

```
typedef struct _CLASSINFOSTRUCT
{
    ULONG                ulStruct;
    PCLASSATTRSTRUCT    pClassAttrStruct;
    USHORT               usNbrAttrIds;
    USHORT               usMaxVersions;
    USHORT               usIndexClass;
    USHORT               usViewID;
    CHAR                 szACLName;
    CHAR                 achLanguageCode;
    CHAR                 szClassName;
    CHAR                 szDescription;
    CHAR                 szCollectionName;
    CHAR                 szStoreSite;
} CLASSINFOSTRUCT, *PCLASSINFOSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>pClassAttrStruct</i>	PCLASSATTRSTRUCT — output A pointer to an array of class attribute structures.
<i>usNbrAttrIds</i>	USHORT — output The number of attribute IDs in the CLASSATTRSTRUCT array. For classes with no attributes, this value is 0, and the <i>pClassAttrStruct</i> field contains the value NULL.
<i>usMaxVersions</i>	USHORT — output Not supported.
<i>usIndexClass</i>	USHORT — output An index class identifier.
<i>usViewID</i>	USHORT — output The ID of an existing index class view. Content Manager for iSeries supports only a single view, with the same identifier as the index class.
<i>szACLName</i>	CHAR[SIM_ACCESS_LIST_NAME_LENGTH+1] — output The name of the access list (ACL) for the index class.
<i>achLanguageCode</i>	CHAR[SIM_LANGUAGE_CODE_LENGTH+1] — output

CLASSINFOSTRUCT

	The 3-character national language code for this index class name or view name. The values for language codes are described in the <i>IBM National Language Design Guide: National Language Support Reference Manual, Volume 2</i> .
<i>szClassName</i>	CHAR[SIM_CLASS_NAME_LENGTH+1] — output The name of the index class or view, expressed in the specified language.
<i>szDescription</i>	CHAR[SIM_DESCRIPTION_LENGTH+1] — output Not supported.
<i>szCollectionName</i>	CHAR[SIM_COLLECTION_NAME_LENGTH+1] — output The default collection for new objects in the specified index class. For a view, this is the same value as for the index class that is associated with the view. It is valid for a view only on the SimLibGetClassInfo function.
<i>szStoreSite</i>	CHAR[SIM_SERVER_NAME_LENGTH+1] — output Not supported.

CONTENTCLASSINFO (Content Class Information Structure)

This information structure provides the data you need to create and modify a content class. It consists of the following:

```
typedef struct _CONTENTCLASSINFO
{
    ULONG                ulStruct;
    USHORT               usContentClsID;
    CHAR                 szContentClsName;
    CHAR                 szContentClsDesc;
} CONTENTCLASSINFO, *PCONTENTCLASSINFO;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>usContentClsID</i>	USHORT — output An unique content class ID that Content Manager for iSeries generates.
<i>szContentClsName</i>	CHAR[9] — output The name of the content class.
<i>szContentClsDesc</i>	CHAR[41] — output The description of the content class.

HOBJ (Handle to Query Stored Object)

This handle identifies the stored object to query. This is actually a pointer to a data structure that consists of:

```
typedef struct _OBJSTRUCT
{
    ULONG                ulStruct;
    ULONG                ulPart;
    SHORT               sVersion;
    ITEMID              szItemID;
    UCHAR               chRepType;
    UCHAR               chReserved;
} OBJ, *HOBJ;
```

Fields

<i>ulStruct</i>	ULONG — input/output The length of the structure in bytes, including the length of this field.
<i>ulPart</i>	ULONG — input/output The part number of the object. Only positive values are valid.
<i>sVersion</i>	SHORT — input Not supported.
<i>szItemID</i>	ITEMID — input/output The item ID of the object.
<i>chRepType</i>	UCHAR[<i>SIM_REP_TYPE</i>] — input/output Not supported.
<i>chReserved</i>	UCHAR[<i>SIM_OBJ_RESERVED_LENGTH</i>] — input Reserved.

ICVIEWSTRUCT (Index Class View Information Structure)

This data structure provides information about the index class or index class view information structure. It consists of the following:

```
typedef struct _ICVIEWSTRUCT
{
    ULONG                ulStruct;
    struct _ICVIEWSTRUCT *pNextView;
    PATRLISTSTRUCT      pAttr;
    USHORT              usIndexClass;
    USHORT              usViewId;
    USHORT              usNumAttributes;
} ICVIEWSTRUCT, *PICVIEWSTRUCT;
```

ICVIEWSTRUCT

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>pNextView</i>	struct _ICVIEWSTRUCT * — output The pointer to the next field in the linked list of view information for the item. Each field in this list is an ICVIEWSTRUCT data structure. For Content Manager for iSeries, this pointer always contains the value NULL.
<i>pAttr</i>	PATRLISTSTRUCT — output The pointer to an array of PATRLISTSTRUCT data structures. Each data structure contains either the system-defined or the user-defined attribute ID of the current view for this item. One data structure in the array specifies one attribute.
<i>usIndexClass</i>	USHORT — output The index class identifier for the item.
<i>usViewId</i>	USHORT — output The ID of an existing index class view. Content Manager for iSeries supports only a single view, with the same identifier as the index class.
<i>usNumAttributes</i>	USHORT — output The number of attribute values that exist for this item. The value of this field matches the number of PATRLISTSTRUCT data structures that the <i>pAttr</i> field points to.

ITEMINFOSTRUCT (Item Information Structure)

This data structure provides the requested item information. It consists of the following:

```
typedef struct _ITEMINFOSTRUCT
{
    ULONG                ulStruct;
    BOOL                fSuspended;
    USHORT               usItemType;
    USHORT               usIndexClass;
    ULONG               ulOpenStatus;
    USHORT               usWipStatus;
    USERID              useridCheckout;
    CHAR                szLabel;
} ITEMINFOSTRUCT, *PITEMINFOSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output
-----------------	----------------

	The length of the structure in bytes, including the length of this field.
<i>fSuspended</i>	<p>BOOL — output</p> <p>Not supported.</p>
<i>usItemType</i>	<p>USHORT — output</p> <p>The type of items retrieved using the SimLibGetItemInfo function. The valid values are:</p> <p>SIM_DOCUMENT Indicates that the item is a document.</p> <p>SIM_FOLDER Indicates that the item is a folder.</p> <p>SIM_WORKBASKET Indicates that the item is a workbasket.</p> <p>SIM_WORKFLOW Indicates that the item is a process.</p>
<i>usIndexClass</i>	<p>USHORT — output</p> <p>An index class identifier.</p> <p>For the SimLibGetItemInfo function, this value specifies the index class ID for the item you are querying.</p>
<i>ulOpenStatus</i>	<p>ULONG — output</p> <p>Indicator of whether the item is open for update. Together, this parameter and the <i>useridCheckout</i> parameter provide information about who has the item and for what purpose. The valid values are:</p> <p>SIM_ACCESS_READ_WRITE Indicates that you have the item open for update.</p> <p>SIM_ACCESS_UNKNOWN Indicates that you do not have the item open for update.</p>
<i>usWipStatus</i>	<p>USHORT — output</p> <p>The current WIP status of the item. The value of this field indicates whether or not the item is suspended, as well as the workflow status of the item. The OR operator is used to combine one suspension status value with one workflow status value from the following groups:</p> <p>Suspension Status Values Not supported.</p> <p>Workflow Status Values</p> <p>OIM-CURRENT_WORKFLOW_ITEMS Indicates that the item is in a process.</p>

ITEMINFOSTRUCT

OIM_ITEMS_NOT_IN_WORKFLOW

Indicates that the item is not in a process.

useridCheckout

USERIDENT — output

The user ID of the person who has the item checked out. Together, this parameter and the *ulOpenStatus* parameter provide information about who has the item and for what purpose. The valid values are:

Your user ID

Indicates that you have the item checked out permanently and open for update, if *ulOpenStatus* contains the value SIM_ACCESS_READ_WRITE. Otherwise, you have the item checked out permanently but it is not open for update.

Other user ID

Identifies another user who has the item checked out, if *ulOpenStatus* contains SIM_ACCESS_UNKNOWN.

A null string

Indicates that you have the item open for update, if *ulOpenStatus* contains the value SIM_ACCESS_READ_WRITE. Otherwise, the item is not checked out.

szLabel

CHAR[SIM_LABEL_LENGTH+1] — output

A null-terminated string that contains the name or label of the item.

ITEMNAMESTRUCT (Item Name Data Structure)

This data structure provides the name associated with a workbasket or process item.

```
typedef struct_ITEMNAMESTRUCT
{
    ULONG                ulStruct;
    ITEMID               WItemID;
    CHAR                 szIDName;
    ULONG                ulActive;
} ITEMNAMESTRUCT, *PITEMNAMESTRUCT;
```

Fields

ulStruct

ULONG — output

The length of the structure in bytes, including the length of this field.

WItemID

ITEMID — output

The item ID of either the workbasket or the process.

szIDName

CHAR[OIM_ITEMNAME_LENGTH+1] — output

The description of the item .

ulActive

ULONG — output

For Content Manager for iSeries, the status of the workbasket or process.
The valid values are:**SIMWM_ACTIVE**

Indicates the workbasket or process is active.

SIMWM_INACTIVE

Indicates the workbasket or process is marked for deletion.

LIBSEARCHCRITERIASTRUCT (Search Criteria Information Structure)

This data structure provides information about which index class to search and the search expression itself. It consists of the following:

```
typedef struct _LIBSEARCHCRITERIASTRUCT
{
    ULONG                ulStruct;
    ULONG                ulReturnLimit;
    BITS                 fSearch;
    PSZ                  pszSearchString;
    USHORT               usViewID;
    USHORT               usSearchUnused;
} LIBSEARCHCRITERIASTRUCT, *PLIBSEARCHCRITERIASTRUCT;
```

Fields

ulStruct

ULONG — input

The length of the structure in bytes, including the length of this field.

ulReturnLimit

ULONG — input

The maximum number of items that the search returns for the index class you specify. If you specify SIM_SEARCH_ALLVIEWS as the value of the *fSearch* field, the value of this field is the maximum number of items that the search returns per index class from each index class you search. Specify 0 as the value of this field to return all the items that match the search criteria for the index class you specify.*fSearch*

BITS — input

The search modification indicator. The value of this field determines a modification to the search. The valid values are:

SIM_SEARCH_VIEW

Searches only the view specified in the

LIBSEARCHCRITERIASTRUCT

usViewID field. If you specify this value, you must specify the ID of a valid view in the *usViewID* field.

SIM_SEARCH_ALLVIEWS

Searches all the appropriate current views, not just one view. If you specify this value, you must specify 0 as the value of the *usViewID* field. You can specify this value in only one of the data structures in an array of search criteria.

If you specify this value, the **SimLibSearch** function automatically searches only the views that contain the attributes you specify in the expression within the *pszSearchString* field.

pszSearchString

PSZ — input

A pointer to a null-terminated string. This field contains one or more expressions. Each expression describes the search conditions on an attribute. Use logical operators to combine expressions for the search. You can use an unlimited number of levels and parentheses. See Guidelines for Search Expressions following this list.

usViewID

USHORT — input

The ID of an existing index class.

usSearchUnused

USHORT — input

Reserved field.

Restriction: The **SimLibSearch** function does not use this value.

Guidelines for Search Expressions

See Appendix A, “Guidelines for Search Expressions,” on page 291.

LIBSESSIONINFOSTRUCT (Library Session Information Structure)

This data structure provides information about the current library session that you specify as the value of the HSESSION parameter, when you use the **SimLibLogon** function to start the current session. It consists of the following:

```
typedef struct _LIBSESSIONINFOSTRUCT
{
    ULONG                ulStruct;
    SESSION_P           pSession;
    CHAR                szDBName;
    CHAR                szApplicationName;
    PATRON_ID           szUserIDSession;
} LIBSESSIONINFOSTRUCT, *PLIBSESSIONINFOSTRUCT;
```


Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>pSession</i>	SESSION_P — output The handle to a library client session.
<i>szDBName</i>	CHAR[RS_STORE_ID_LENGTH+1] — output Not supported.
<i>szApplicationName</i>	CHAR[RS_STORE_ID_LENGTH+1] — output Not supported.
<i>szUserIDSession</i>	PATRON_ID — output The current user ID for the session.

NAMESTRUCT (Name Data Structure)

This data structure provides the name associated with an attribute or index class view code. It consists of the following:

```
typedef struct _NAMESTRUCT
{
    ULONG                ulStruct;
    USHORT              usID;
    CHAR                 szName;
    CHAR                 szDescription;
} NAMESTRUCT, *PNAMESTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>usID</i>	USHORT — output The ID of a valid attribute, an index class, or an index class view.
<i>szName</i>	CHAR[SIM_CLASS_NAME_LENGTH+1] — output The name of the index class or view in the current language.
<i>szDescription</i>	CHAR[SIM_DESCRIPTION_LENGTH+1] — output Not supported.

OBJINFOSTRUCT (Object Information Structure)

This data structure provides storage information about the object. It consists of the following:

OBJINFOSTRUCT

```
typedef struct _OBJINFOSTRUCT
{
    ULONG                ulStruct;
    ULONG                ulObjSize;
    LONG                 lSMSRetention;
    LONG                 lEstimateRetrieveTime;
    ULONG                ulAvail;
    ULONG                ulObjConCls;
    USHORT               usPageNum;
    TIMESTAMP            tsCreate;
    TIMESTAMP            tsExpiration;
    TIMESTAMP            tsLastRef;
    TIMESTAMP            tsModify;
    TIMESTAMP            tsEnterSG;
    TIMESTAMP            tsEnterSC;
    CHAR                 szCollectionName;
    CHAR                 szObjectName;
    CHAR                 szMgtCls;
    CHAR                 szStgCls;
    CHAR                 szDataCls;
    CHAR                 szStoreSite;
} OBJINFOSTRUCT, *POBJINFOSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>ulObjSize</i>	ULONG — output The total size of the object in bytes.
<i>lSMSRetention</i>	LONG — output Not supported.
<i>lEstimateRetrieveTime</i>	LONG — output Not supported.
<i>ulAvail</i>	ULONG — output Not supported.
<i>ulObjConCls</i>	ULONG — output The object content class of the object you query. The value SIM_CC_UNKNOWN indicates the undefined content class.
<i>usPageNum</i>	USHORT — output Not supported.
<i>tsCreate</i>	TIMESTAMP — output The date and time that the item or object was created.
<i>tsExpiration</i>	TIMESTAMP — output

	Not supported.
<i>tsLastRef</i>	TIMESTAMP — output
	Not supported.
<i>tsModify</i>	TIMESTAMP — output
	The date and time that the item or object was last modified.
<i>tsEnterSG</i>	TIMESTAMP — output
	Not supported.
<i>tsEnterSC</i>	TIMESTAMP — output
	Not supported.
<i>szCollectionName</i>	CHAR[MAXCOLNMSZ] — input
	Not supported.
<i>szObjectName</i>	CHAR[MAXOBJNMSZ] — input
	Not supported.
<i>szMgtCls</i>	CHAR[MAXMGTCLSNMSZ] — output
	Not supported.
<i>szStgCls</i>	CHAR[MAXSTGCLSNMSZ] — output
	Not supported.
<i>szDataCls</i>	CHAR[MAXDATACLSNMSZ] — output
	Not supported.
<i>szStoreSite</i>	CHAR[MAXSTRSITENMSZ] — output
	Not supported.

RCSTRUCT (Return Code Information Structure)

This data structure provides programming-interface function return code and data information. It consists of the following:

```
typedef struct _RCSTRUCT
{
    ULONG                ulStruct;
    ULONG                ulRC;
    USHORT               usReserved;
    USHORT               usParam;
    ULONG                ulParam1;
    ULONG                ulParam2;
#ifdef _OS400_
    PVOID                pParam1;
    PVOID                pParam2;
#endif
    ULONG                ulExtRC;
    ULONG                ulExtReason;
    PVOID                pApplData;
    ULONG                ulApplData;
    ULONG                ulReserved;
};
```

RCSTRUCT

```
HERR                hErrLog;  
  
} RCSTRUCT, *PRCSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>ulRC</i>	ULONG — output The function return code.
<i>usReserved</i>	USHORT — output Not supported.
<i>usParam</i>	USHORT — output A field that indicates whether the <i>ulParam1</i> field contains a pointer to a data area. The value 1 indicates that this is the case. Otherwise, this field contains the value 0.
<i>ulParam1</i>	ULONG — output A value or a pointer to either a data structure or an array of data structures.
<i>ulParam2</i>	ULONG — output A field that indicates the number of data structures in the array if the <i>ulParam1</i> field contains a pointer to an array of data structures.
<i>pParam1</i>	PVOID — output A pointer used in place of <i>ulParam1</i> when the function is executed on the server.
<i>pParam2</i>	PVOID — output A pointer used in place of <i>ulParam2</i> when the function is executed on the server.
<i>ulExtRC</i>	ULONG — output A return code from other components that Content Manager for iSeries called directly or indirectly.
<i>ulExtReason</i>	ULONG — output Not supported.
<i>pApplData</i>	PVOID — output A PVOID data field that your application can use to contain application data. Content Manager for iSeries does not use this data field. The value is preserved by the programming interface function and returned. For example, your application might use this field to point to a data structure, one that your application creates prior to using a function

that requires the data. The function could use the data in that structure to process a user exit.

ulApplData

ULONG — output

A ULONG data field that your application can use to contain application data. Content Manager for iSeries does not use this data field. The value is preserved by the programming interface function and returned. For example, your application might use this field to point to a data structure, one that your application creates prior to using a function that requires the data. The function could use the data in that structure to process a user exit.

ulReserved

ULONG — output

Not supported.

hErrLog

HERR — output

Not supported.

SERVERINFOSTRUCT (Server Information Structure)

The structure contains information about a server defined to the system. This data structure is returned to the application that called it. It consists of the following:

```
typedef struct _SERVERINFOSTRUCT
{
    ULONG                ulStruct;
    CHAR                szServerName;
    CHAR                szServerType;
} SERVERINFOSTRUCT, *PSERVERINFOSTRUCT;
```

Fields

ulStruct

ULONG — input

The length of the structure in bytes, including the length of this field.

szServerName

CHAR[SERVERNAME LENG+1]— output

The name of the Content Manager for iSeriesserver.

szServerType

CHAR[SEVERTYPE LENG+1]— output

The server type. Current server types include the following:

Server Type	Explanation
'FRNCACHE'	List manager cache
'FRNREXE'	Remote utility server
'FRNCS'	Configuration server
'FRNOSADM'	System-managed storage server
'FRNOLM'	List manager server

SMS (System-Managed Storage Pointer)

The pointer to the system-managed storage (SMS) data structure for an object. This data structure provides the information necessary to support the SMS for an object on a variety of object servers. This is a pointer to a data structure that consists of the following:

```
typedef struct _SMS
{
    ULONG                ulStruct;
    LONG                lSMSRetention;
    CHAR                szCollectionName;
    CHAR                szObjectName;
    CHAR                szMgtCls;
    CHAR                szStgCls;
    CHAR                szDataCls;
    CHAR                szStoreSite;
    CHAR                szStoreHint;
} SMS, *PSMS;
```

Fields

<i>ulStruct</i>	ULONG — input The length of the structure in bytes, including the length of this field.
<i>lSMSRetention</i>	LONG — input The period in days that Content Manager for iSeries retains the object in system-managed storage. The valid values range from 1 to 999 999 999.
<i>szCollectionName</i>	CHAR[MAXCOLNMSZ] — input The ASCIIZ user-defined collection name. The value of this field references a zero-terminated string in client data space, containing a user-defined number of significant characters. This character string provides a meaningful name for the collection being created. If you do not require a collection name, specify the value NULL. After an object has been assigned to a collection on an object server, you cannot change the collection assignment.
<i>szObjectName</i>	CHAR[MAXOBJNMSZ] — input Not supported.
<i>szMgtCls</i>	CHAR[MAXMGTCLSNMSZ] — input Not supported.
<i>szStgCls</i>	CHAR[MAXSTGCLSNMSZ] — input Not supported.
<i>szDataCls</i>	CHAR[MAXDATACLSNMSZ] — input

	Not supported.
<i>szStoreSite</i>	CHAR[MAXSTRSITENMSZ] — input The name of the object server in which the object is stored.
<i>szStoreHint</i>	CHAR[MAXSTGHINTNMSZ] — input Not supported.

SNAPSHOTSTRUCT (Snapshot Information Structure)

This data structure provides the view, attribute, and work management information for an item at a specific point in time. It consists of the following:

```
typedef struct _SNAPSHOTSTRUCT
{
    ULONG                ulStruct;
    PWMSNAPSHOTSTRUCT   pWmSnapshot;
    USHORT              usNumWmSnapshots;
    PICVIEWSTRUCT       pICView;
    USHORT              usNumViews;
    USHORT              usItemType;
    ULONG               ulOpenStatus;
    ITEMID              szItemID;
    USERID              useridCheckout;
    TIMESTAMP           tsCreate;
    TIMESTAMP           tsModify;
} SNAPSHOTSTRUCT, *PSNAPSHOTSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>pWmSnapshot</i>	PWMSNAPSHOTSTRUCT — output The pointer to the workflow information data structure of the type WMSNAPSHOTSTRUCT. The SimLibGetItemSnapshot function returns this structure when the you specify the value of the <i>fReadAttrInd</i> input parameter as SIM_WORK_ATTR. Otherwise, this field contains the value NULL. Content Manager for iSeries supports the existence of an item in more than one workbasket, so it could be an array of workflow information for an item.
<i>usNumWmSnapshots</i>	USHORT — input The number of elements in the array of WMSNAPSHOTSTRUCT that <i>pWmSnapshot</i> points to.
<i>pICView</i>	PICVIEWSTRUCT — output

SNAPSHOTSTRUCT

	<p>The pointer to a linked list of view information for the item, where each element of the list is of the data type ICVIEWSTRUCT. If the item is not associated with any index class, or you do not retrieve system attributes, this pointer contains the value NULL.</p> <p>Currently in Content Manager for iSeries, if the item is associated with an index class, there is only one element in the linked list containing information about the current index class view for the item. If the item is not associated with any index class, this pointer contains the value NULL.</p>
<i>usNumViews</i>	<p>USHORT — output</p> <p>The number of elements in the linked list pointed to by the <i>pICView</i> field in the SNAPSHOTSTRUCT data structure.</p> <p>Currently in Content Manager for iSeries, if the item is associated with an index class, this field contains the value 1. This value indicates that the linked list of elements of the data type ICVIEWSTRUCT contains one element with information pertaining to the current index class view for the item. If the item is not associated with an index class, this field contains the value 0. In this case, however, the <i>pICView</i> pointer is still valid if you retrieve system attributes.</p>
<i>usItemType</i>	<p>USHORT — output</p> <p>The type of items retrieved using the SimLibGetItemSnapshot function. The valid values are:</p> <p>SIM_DOCUMENT Indicates that the item is a document.</p> <p>SIM_FOLDER Indicates that the item is a folder.</p>
<i>ulOpenStatus</i>	<p>ULONG — output</p> <p>Indicator of whether the item is open for update. Together, this parameter and the <i>useridCheckout</i> parameter provide information about who has the item and for what purpose. The valid values are:</p> <p>SIM_ACCESS_READ_WRITE Indicates that you have the item open for update.</p> <p>SIM_ACCESS_UNKNOWN Indicates that you do not have the item open for update.</p>
<i>szItemID</i>	<p>ITEMID — output</p> <p>An item ID.</p>
<i>useridCheckout</i>	<p>USERIDENT — output</p>

The user ID of the person who has the item checked out. Together, this parameter and the *ulOpenStatus* parameter provide information about who has the item and for what purpose. The valid values are:

Your user ID

Indicates that you have the item checked out permanently and open for update, if *ulOpenStatus* contains the value `SIM_ACCESS_READ_WRITE`. Otherwise, you have the item checked out permanently but it is not open for update.

Other user ID

Identifies another user who has the item checked out, if *ulOpenStatus* contains `SIM_ACCESS_UNKNOWN`.

A null string

Indicates that you have the item open for update, if *ulOpenStatus* contains the value `SIM_ACCESS_READ_WRITE`. Otherwise, the item is not checked out.

<i>tsCreate</i>	TIMESTAMP — output The date and time that the item or object was created.
<i>tsModify</i>	TIMESTAMP — output The date and time that the item or object was last modified.

TOCENTRYSTRUCT (Table of Contents Entry Data Structure)

This data structure provides information describing an entry in a list of the documents and folders contained in the specific folder or workbasket. It consists of the following:

```
typedef struct _TOCENTRYSTRUCT
{
    ULONG                ulStruct;
    USHORT               usItemStatus;
    USHORT               usIndexClass;
    USHORT               usItemType;
    ITEMID               szItemID;
    TIMESTAMP            tsItemChanged;
} TOCENTRYSTRUCT, *PTOCENTRYSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — input The length of the structure in bytes, including the length of this field.
<i>usItemStatus</i>	USHORT — input

TOCENTRYSTRUCT

	The status of the entry after the update. The valid values are: <ul style="list-style-type: none">• 0 (unmodified)• SIM_TOC_ADD• SIM_TOC_MODIFIED• SIM_TOC_DELETE
<i>usIndexClass</i>	USHORT — input An index class identifier.
<i>usItemType</i>	USHORT — input The type of items retrieved using the SimLibGetTOC function. The valid values are: SIM_DOCUMENT Indicates that the item is a document. SIM_FOLDER Indicates that the item is a folder.
<i>szItemID</i>	ITEMID — input An item ID.
<i>tsItemChanged</i>	TIMESTAMP — input The timestamp of the item as stored in the library server.

USERACCESSSTRUCT (User Access Data Structure)

This data structure provides information describing the user who has checked out the referenced item. It consists of the following:

```
typedef struct _USERACCESSSTRUCT
{
    ULONG                ulStruct;
    ULONG                ulAccessLevel;
    USERIDENT           useridCheckout;
    ITEMID               szItemID;
} USERACCESSSTRUCT, *PUSERACCESSSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — output The length of the structure in bytes, including the length of this field.
<i>ulAccessLevel</i>	ULONG — output Not supported.
<i>useridCheckout</i>	USERIDENT — output The user ID of the person who checked out this item. If the item is not currently checked out, this field contains the value NULL.
<i>szItemID</i>	ITEMID — output

An item ID.

USERLOGONINFOSTRUCT (User Logon Information Structure)

This data structure provides information about the user's session. It consists of the following:

```
typedef struct _USERLOGONINFOSTRUCT
{
    ULONG                ulStruct;
    ULONG                ulUserType;
    ULONG                ulUserCCSID;
    PSZ                  pszUserDescription;
    CHAR                 szUserLanguage;
    CHAR                 szSessionType;
    TIMESTAMP            tsPasswordExpire;
    CHAR                 szPrivString;
} USERLOGONINFOSTRUCT, *PUSERLOGONINFOSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — input The length of the structure in bytes, including the length of this field.
<i>ulUserType</i>	ULONG — input Not supported.
<i>ulUserCCSID</i>	ULONG — input Not supported.
<i>pszUserDescription</i>	PSZ — input Not supported.
<i>szUserLanguage</i>	CHAR[SIM_LANGUAGE_CODE_LENGTH+1] — input A fixed-length character array that indicates the language that this user prefers for dialogs and messages. The valid value is a standard IBM 3-character language code. The values for language codes are described in the <i>IBM National Language Design Guide: National Language Support Reference Manual Volume 2</i>
<i>szSessionType</i>	CHAR[SIM_SESSION_TYPE_LENGTH+1] — input The type of logon session. The only valid value for this field is Ip2.
<i>tsPasswordExpire</i>	TIMESTAMP — input The date when the current password expires.
<i>szPrivString</i>	CHAR[SIM_PRIVSTRING_LENGTH+1] — input A null-terminated character string that represents the privilege vector for the user. This string

consists of ASCII zeros and ones that correspond to the zeros and ones in the user's corresponding privilege vector.

WMACTIONLISTFUNCSTRUCT (Action List Function Structure)

This data structure provides information about an action that is defined with an action list.

```
typedef struct _WMACTIONLISTFUNCSTRUCT
{
    ULONG                ulFuncNumber;
    ULONG                ulActionType;
    ULONG                ulFuncCode;
    CHAR                 szFuncPrompt;
    CHAR                 szAction;
    CHAR                 szIcon;
    CHAR                 szShortcut;
    CHAR                 szExitFuncName;
    CHAR                 szExitDLLName;
} WMACTIONLISTFUNCSTRUCT, *PWMACTIONLISTFUNCSTRUCT;
```

Fields

ulFuncNumber

ULONG — output

The sequence number of the action within the action list.

ulActionType

ULONG — output

Indicates whether an action is applicable for documents, folders, or both item types. The valid values are:

SIMWM_ACTION_DOCUMENT

The action is associated with document items.

SIMWM_ACTION_FOLDER

The action is associated with folder items.

SIMWM_ACTION_BOTH

The action is associated with both folder and document items.

ulFuncCode

ULONG — output

The value that uniquely identifies an action.

szFuncPrompt

CHAR[SIMWM_AL_PROMPT+1] — output

The text prompt associated with this action.

szAction

CHAR[SIMWM_AL_ACTION+1] — output

The value to be assigned to the SIMWM_ACTION variable when this action is selected.

szIcon

CHAR[SIMWM_AL_ICON+1] — output

Icon associated with this action.

szShortcut

CHAR[SIMWM_AL_SHORTCUT+1] — output

The keyboard shortcut associated with this action.

szExitFuncName

CHAR[OIM_WB_FUNCTION_LENGTH+1] — output

If this is a user-defined action, this field contains the name of the user exit function to be run.

szExitDLLName

CHAR[OIM_WB_DLL_LENGTH+1] — output

If this is a user-defined action, this field contains the name of the dynamic link library which contains the function *szExitFuncName*.

WMACTIONLISTINFOSTRUCT (Action List Data Structure)

This data structure provides all of the information associated with an action list definition.

```
typedef struct _WMACTIONLISTINFOSTRUCT
```

```
{
    ULONG                ulStruct;
    CHAR                 szActionListName;
    TIMESTAMP            tsALCreate;
    TIMESTAMP            tsALModify;
    CHAR                 szDescription;
    ULONG                ulALNumFunctions;
    PWMACTIONLISTFUNCSTRUCT pALFunctions;
}
```

```
}WMACTIONLISTINFOSTRUCT, *PWMACTIONLISTINFOSTRUCT;
```

Fields

ulStruct

ULONG — output

The length of the structure in bytes, including the length of this field.

szActionListName

CHAR[SIMWM_ACTION_LENGTH+1] — output

The name of the action list.

tsALCreate

TIMESTAMP — output

The date and time the action list was created.

tsALModify

TIMESTAMP — output

The date and time the action list was last modified.

szDescription

CHAR[SIMWM_AL_DESCRIPTION+1] — output

Description of the action list.

WMACTIONLISTINFOSTRUCT

ulALNumFunctions

ULONG — output

The number of functions associated with this action list.

pALFunctions

PWMACTIONLISTFUNCSTRUCT — output

Pointer to the list of functions associated with this action list.

WMHISTLOGENTRYSTRUCT (WMEvent History Structure)

This data structure provides the history for a work package in an array of the history log entries for the work package.

```
typedef struct _HISTLOGENTRY
```

```
{  
    CHAR                szEventID;  
    TIMESTAMP          tsCreated;  
    CHAR                szProcess;  
    CHAR                szLocation;  
    USERIDENT          szUser;  
    CHAR                szEventData;  
};
```

```
}WMHISTLOGENTRYSTRUCT, *PWMHISTLOGENTRY;
```

Fields

szEventID

CHAR[7] — output

The seven-character message ID.

tsCreated

TIMESTAMP — output

The date and time of the event.

szProcessID

CHAR[SIMWM_PROCESS_NAME_LENGTH+1] — output

The WorkFlow process name.

szLocation

CHAR[SIMWM_LOC_NAME_LENGTH+1] — output

The WorkFlow location name.

szUser USERIDENT — output

The user ID.

szEventData

CHAR[256] — output

The text description associated with the event.

WMLOCATIONINFOSTRUCT (Work Process Location Information Structure)

This data structure provides information associated with each location within a process.

```
typedef struct _WMLOCATIONINFOSTRUCT
{
    ULONG                ulType;
    CHAR                szLocation;
    CHAR                szDescription;
    ULONG                ulActive;
} WMLOCATIONINFOSTRUCT, *PWMLOCATIONINFOSTRUCT;
```

Fields

ulType ULONG — output

Indicates whether the returned information is a workbasket or a collection point. The valid values are:

SIM_WORKBASKET

Indicates the location is a workbasket.

SIM_COLLECTION_POINT

Indicates the location is a collection point.

szLocation

CHAR[SIMWM_LOC_NAME_LENGTH+1] — output

The workbasket or collection point identifier.

szDescription

CHAR[SIMWM_LOC_DESC_LENGTH+1] — output

The text description associated with the location.

ulActive

ULONG — output

Not supported.

WMPROCESSINFOSTRUCT (Process Information Data Structure)

This data structure provides information about a specific process.

```
typedef struct _WMPROCESSINFOSTRUCT
{
    ULONG                ulStruct;
    CHAR                szProcessID;
    CHAR                szProcessDescription;
    CHAR                chAccessListName;
    USHORT              usHistoryLogDisposition;
    ULONG                ulNbrItemsInProgress;
    ULONG                ulNbrLocations;
    UCHAR               szPrioString;
    PWMLOCATIONINFOSTRUCT pLocations;
} WMPROCESSINFOSTRUCT, *PWMPROCESSINFOSTRUCT;
```

WMPROCESSINFOSTRUCT

Fields

ulStruct

ULONG — output

The length of the structure in bytes, including the length of this field.

szProcessID

CHAR[SIM_PROCESS_NAME_LENGTH+1] — output

The process identifier.

szProcessDescription;

CHAR[SIM_DESCRIPTION_LENGTH+1] — output

The text description associated with the process.

chAccessListName

CHAR[ACCESS_LIST_NAME_SIZE+1] — output

The name of the access list for the process.

usHistoryLogDisposition

USHORT — output

Not supported

ulNbrItemsInProcess

ULONG — output

The number of work packages on the process.

ulNbrLocations

ULONG — output

The number of unique locations defined within the process.

szPrivString

UCHAR[SIM_PRIVSTRING_LENGTH+1] — output

The evaluated privilege string for the user with respect to the process.

pLocations

PWMLOCATIONINFOSTRUCT — output

The pointer to the array location information data structures of the type WMLOCATIONINFOSTRUCT.

WMSNAPSHOTSTRUCT (Work Management Information Structure)

This data structure provides workflow information associated with an item. It consists of the following:

```
typedef struct _WMSNAPSHOTSTRUCT
```

```
{  
    ULONG                ulStruct;  
    USHORT               usWIPStatus;  
    USHORT               usReleaseType;  
    USHORT               usPriority;  
    ITEMID               szWorkFlowID;  
    TIMESTAMP            tsWFEntry;  
    TIMESTAMP            tsEnteredWB;  
    ITEMID               szWorkBasketID;  
};
```


WMSNAPSHOTSTRUCT

<i>szWorkBasketID</i>	ITEMID — output The workbasket identifier that this item is assigned to.
<i>ulWorkPackageID</i>	ULONG — output Identifier of the work package that represents the work being done, such as the document being routed.
<i>ulInstanceID</i>	ULONG — output Identifier of the work package instance that distinguishes one parallel path from another within the process.
<i>ulLocationType</i>	ULONG — output Indicator of whether the location is a workbasket or collection point. The valid values are: SIMWM_WORKBASKET Indicates the location is a workbasket. SIMWM_COLLECTION POINT Indicates the location is a collection point.
<i>szLocation</i>	CHAR [SIMWM_LOC_NAME_LENGTH+1] — output The workbasket or collection point identifier of the location where the work package resides.
<i>tsEnteredLocation</i>	TIMESTAMP — output The date and time the item entered location.
<i>szOverrideAction</i>	CHAR[SIMWM_ACTION_LENGTH+1] — output Action list associated with this work package. This action list will override the default action list defined by the workbasket definition.

WMSUSPENDSTRUCT (Suspend Work Package Data Structure)

This data structure provides data regarding the release criteria of a suspended item. It consists of the following:

```
typedef struct _WMSUSPENDSTRUCT
{
    ULONG                ulStruct;
    USHORT               usReleaseType;
    TIMESTAMP            tsExpDateTime;
    CHAR                 szExpWB;
    CHAR                 szReadyWB;
    USHORT               usNumAwaitedClasses;
    USHORT               usAwaitedClasses;
} WMSUSPENDSTRUCT, *PWMSUSPENDSTRUCT;
```

Fields*ulStruct*

ULONG — input

The length of the structure in bytes, including the length of this field.

usReleaseType

ULONG — input

The type of criteria in effect for releasing an item from suspension. The valid values are:

SIMWM_SUSPEND_TIMESuspend until the expiration time specified by *tsExpDateTime*.**SIMWM_SUSPEND_ANY_CLASS**Suspend until a folder receives an item of any index class listed in *ausAwaitedClasses*. A preset time is also required in *tsExpDateTime*.**SIMWM_SUSPEND_ALL_CLASS**Suspend until a folder receives an item from each class listed in *ausAwaitedClasses*. A preset time is also required in *tsExpDateTime*.*tsExpDateTime*

TIMESTAMP — input

The date and time to release the work package from suspension.

usNumAwaitedClasses

USHORT — input

The number of index class entries in the *ausAwaitedClasses* array.If SIM_INDEX_ANY is entered for *ausAwaitedClasses*, this number must be one (1).*ausAwaitedClasses*

CHAR[SIMWM_MAX_AWAIT_CLASSES] — input

An array of one to eight index classes that you can specify as suspension criteria for a particular folder work package. The index class SIM_INDEX_ANY may be specified to suspend a folder work package until the arrival of an item of any index class.

szExpWB

CHAR[SIM_ITEM_ID_LENGTH+1] — input

The identifier of the workbasket to send the suspended work package to if the expiration time criteria are satisfied. If SIMWM_NEXT is specified, the work package will be continued to the next step of a process.

szReadyWB

CHAR[SIM_ITEM_ID_LENGTH+1] — input

The identifier of a workbasket to send the suspended folder work package to if the suspension criteria are satisfied by adding one of the items of a specified index class to the folder item. If SIMWM_NEXT is specified, the work package will be continued to the next step of a process.

WMVARSTRUCT (Work Package Variable Data Structure)

This data structure contains the identifier and associated value of a system or user-defined work package variable. It consists of the following:

WMVARSTRUCT

```
typedef struct _WMVARSTRUCT
{
    ULONG                ulStruct;
    CHAR                 szVarName;
    CHAR                 szVarValue;
} WMVARSTRUCT, *PWMVARSTRUCT;
```

Fields

<i>ulStruct</i>	ULONG — input/output The length of the structure in bytes, including the length of this field.
<i>szVarName</i>	CHAR [SIMWWM_VAR_NAME_LENGTH+1] — input/output The name of the variable. The following constants represent the system variable names: SIMWWM_ITEMID Item being routed. SIMWWM_INDEX_CLASS Index class of the item. SIMWWM_PRIORITY Priority of the work package. SIMWWM_ACTION Action selected by the user.
<i>szVarValue</i>	CHAR[SIMWWM_VAR_NAME_LENGTH+1] — input/output Pointer to a string which contains the value of the variable.

WORKBASKETINFOSTRUCT (Workbasket Information Data Structure)

This data structure provides the information used to create and modify a workbasket. It consists of the following:

```
typedef struct _WORKBASKETINFOSTRUCT
{
    ULONG                ulStruct;
    CHAR                 szWorkBasketName;
    CHAR                 chAccessListName;
    USHORT               usWBLoadLimit;
    BOOL                 bRemoveAfterIndex;
    BOOL                 bSystemCntl;
    CHAR                 szUserFunName;
    CHAR                 szUserDLLName;
    UCHAR               szWorkBasketPrioString;
    ULONG               ulItemStatusFlag;
    CHAR                 szDefaultAction;
```

```

USHORT          usWorkbasketType;
CHAR            szEntryFunName;
CHAR            szEntryDLLName;
CHAR            szExitFunName;
CHAR            szExitDLLName;
CHAR            szUserDefWBExitFunName;
CHAR            szUserDefWBExitDLLName;

} WORKBASKETINFOSTRUCT, *PWORKBASKETINFOSTRUCT;

```

Fields

<i>ulStruct</i>	<p>ULONG — output</p> <p>The length of the structure in bytes, including the length of this field.</p>
<i>szWorkBasketName</i>	<p>CHAR[OIM_WB_NAME_LENGTH+1] — output</p> <p>The name of the workbasket.</p>
<i>chAccessListName</i>	<p>CHAR[ACCESS_LIST_NAME_SIZE+1] — output</p> <p>The name of the access list for the workbasket.</p>
<i>usWBLoadLimit</i>	<p>USHORT — output</p> <p>The workbasket overload limit.</p> <p>If you try to add an item to the workbasket and the number of items would exceed this limit, the item is not added. However, when you are adding the item you can override this limit and add the item anyway.</p>
<i>bRemoveAfterIndex</i>	<p>BOOL — output</p> <p>A flag that indicates whether the system removes the item from the workbasket after indexing. The valid values are:</p> <p>TRUE Removes the item from this workbasket after it has been indexed.</p> <p>FALSE Does not remove the item from this workbasket after it has been indexed.</p>
<i>bSystemCntl</i>	<p>BOOL — output</p> <p>A flag that indicates whether the system controls item priority within the workbasket. The valid values are:</p> <p>TRUE Indicates that this is a system-assigned workbasket. The system provides the user with the next item in the workbasket when requested. The priority or date of the work package and the order defined for the workbasket—LIFO, FIFO, or priority—determines the order.</p>

WORKBASKETINFOSTRUCT

	FALSE Indicates that this is not a system-assigned workbasket. The user can choose any item in the workbasket.
<i>szUserFunName</i>	CHAR[OIM_WB_FUNCTION_LENGTH+1] — output The name of the user exit function to call when the workbasket's overload trigger exceeds the limit specified as the value of the <i>usWBLoadLimit</i> field. The DLL and function name are for use by your application.
<i>szUserDLLName</i>	CHAR[OIM_WB_DLL_LENGTH+1] — output The name of a DLL that contains the user exit function. The DLL and function name are for use by your application.
<i>szWorkBasketPrivString</i>	UCHAR[SIM_PRIVSTRING_LENGTH+1] — output The evaluated privilege string for the user with respect to the workbasket.
<i>ullItemStatusFlag</i>	ULONG – output Workbasket status flag. The valid values are: SIMWM_ACTIVE Indicates the workbasket is active. SIMWM_INACTIVE Indicates the workbasket is marked for deletion.
<i>szDefaultAction</i>	CHAR(SIMWM_ACTION_LENGTH+1) — output The default action list associated with this workbasket.
<i>usWorkbasketType</i>	USHORT — output The workbasket type. A value of 50-99 represents a user-defined workbasket.
<i>szEntryFunName</i>	CHAR[OIM_WB_FUNCTION_LENGTH+1] — output The user exit function the application will call when an item is selected and opened at the workbasket.
<i>szEntryDLLName</i>	CHAR[OIM_WB_DLL_LENGTH+1] — output The name of the DLL that contains the entry user exit function.
<i>szExitFunName</i>	CHAR[OIM_WB_FUNCTION_LENGTH+1] — output The user exit function the application will call when the user has completed working with an item at the workbasket.
<i>szExitDLLName</i>	CHAR[OIM_WB_DLL_LENGTH+1] — output

WORKBASKETINFOSTRUCT

	The name of the DLL that contains the completion user exit function.
<i>szUserDefWBExitFunName</i>	CHAR[OIM_WB_FUNCTION_LENGTH+1] — output
	The user exit function the application will call when the workbasket is a user-defined workbasket.
<i>szUserDefWBExitDLLName</i>	CHAR[OIM_WB_DLL_LENGTH+1] — output
	The name of the DLL that contains the user-defined workbasket function.

WORKBASKETINFOSTRUCT

Chapter 5. Using the OLE Automation Interface

Using the APIs provided with the Content Manager for iSeries client, you can enable another Windows-based application to log on to Content Manager for iSeries, perform document and folder searches, display table of contents (TOC) lists for search results, folders, or workbaskets, and even display and annotate documents. You accomplish this by using APIs that are based on OLE 2.0 Automation.

Programming with OLE Automation

OLE automation enables an application's command operations to be manipulated from outside that application. The Client for Windows provides OLE automation objects that can be manipulated from programs built using programming environments such as Visual Basic (Version 3.0 or above), Visual C++, and PowerBuilder. To manipulate Client for Windows objects, you need to know the *properties* and *methods* for each object.

Properties

Properties are similar to Visual Basic variables, except they are located inside Client for Windows objects. Just as you can read or write variables, you can set (that is, write) or get (that is, read) properties. Not all properties are read/write properties; some properties are read-only and others are write-only. For example, the *Visible* property of the *Application* object is a read/write property that can be used to find out whether the program is currently visible on the screen. If the value of the property is set to True, the program is currently visible. Setting the value of the *Visible* property to False causes the program to be hidden. On the other hand, the *Name* property of the *Item* object is a read-only property that contains the name by which Content Manager for iSeries refers to the item. An example of a write-only property is the *Application* property *Password*.

Methods

Methods are similar to Visual Basic procedures or function procedures. You can call a method to perform an operation inside the Client for Windows (that is, invoke a command operation). For example, the *OpenWorkbasket* method of the *Application* class displays the Open Workbasket dialog.

Client for Windows Objects

The Client for Windows OLE automation objects are designed according to Microsoft® guidelines. Therefore, as is the case with all applications that follow these guidelines, the Client for Windows has an *Application* object, a *Documents* collection object, and a *Document* object.

In addition, the Client for Windows has an *Items* collection object to manage multiple *Item* objects, and an *Item* object that provides information and interfaces to Content Manager for iSeries items like documents, folders, and workbaskets. Also provided is an *Image* object that holds the document currently open in the image viewer.

An information-only class called *Error* is provided to allow applications to determine what errors have occurred.

Finally, the Client for Windows also supports two helper objects (*EnumDocument* and *EnumItem*) that are needed by Visual Basic to provide object iteration, although they are not created when programming with Visual Basic.

Collection objects are similar to arrays in the sense that they are used to hold other objects. The *Documents* collection holds *Document* objects, while the *Items* collection holds *Item* objects. All OLE automation collection objects share the same methods and properties.

See “Programming Tips” on page 175 for general information about programming with OLE automation and the objects provided with the Client for Windows .

In addition to Visual Basic, the Client for Windows OLE automation API can be used with any programming language or fourth-generation language (4GL) that supports OLE automation.

Application Object

The main Client for Windows object is the *Application* object. Once a program obtains access to the *Application* object, it can get hold of or create all other Client for Windows objects.

The methods and properties of the *Application* object apply to the Client for Windows as a whole. For example, the *Logon* method is invoked to log on to Content Manager for iSeries, and the *Quit* method is invoked to exit the program. Therefore, programs designed to interface with the Client for Windows must first create the *Application* object.

Once the Client for Windows is running, it can be used to interact with Content Manager for iSeries. You can open a TOC, which equates to a *Document* object in OLE automation, you can find or create items (*Item* object), and you can display documents (*Image* object).

Documents Collection

The *Documents* collection can be compared to a queue holding TOCs (folders, search results or workbaskets). The TOCs are represented by *Document* objects.

Most *Document* objects are opened by calling the *Documents* method *OpenTOC*, with an *Item* object as a parameter.

Document Object

Once a *Document* object has been created through the *OpenTOC* method of the *Documents* collection, the object can be displayed, and a number of methods can be executed. For example, you can query any of the items that are currently selected in the *Document* TOC by the user.

Error Object

If an error occurs, all of the pertinent information for the error will be stored in this object, including Content Manager for iSeries return codes.

Image Object

The *Image* object represents a special document. It is the currently visible Content Manager for iSeries document. The *Image* object is opened by calling its *OpenDocument* method with an *Item* object as a parameter.

Items Collection

The *Items* collection object is simply a list of *Items* that are related. For example, the *Document* method *Selections* returns the *Items* collection containing all of the items that are currently selected. It has methods that return a specific *Item* object from the collection, and also has housekeeping methods to delete *Item* objects and the *Items* collection instance.

You can have more than one *Items* collection defined at one time. However, it is your responsibility to keep track of the *Items* collections, because the only way to get an *Items* collection is when it is returned from a method.

Item Object

The *Item* object represents a Content Manager for iSeries item like a document, folder, or workbasket. The *Item* object enables you to display the item (by passing it as a parameter to other objects), query its index class and key fields, re-index it, and perform a number of other actions.

The *Item* object also contains properties describing itself.

Programming Tips

The OLE automation API can be used to integrate the Client for Windows into your application. To integrate the Client for Windows using this API, the development environment for your application must be able to access OLE automation objects. For example, Microsoft Visual Basic, Microsoft Visual C++, and PowerBuilder, as do a number of applications like Microsoft Excel and Microsoft Access.

The following provides programming tips for programming with OLE automation, including information on releasing objects and handling errors.

Releasing Objects

Programming with OLE automation requires paying attention to object release; programs that allocate objects are responsible for freeing the objects after use. For example, a Client for Windows object is created in Visual Basic as follows:

```
Dim MyItem As Object  
Set MyItem = MyApp.GetWorkbasket("To be indexed")
```

In this operation, the Client for Windows allocates memory to hold the *Item* object and returns a pointer to the object. The pointer is stored in the *MyItem* variable.

To release the *Item* object, use a statement as follows:

```
Set MyItem = Nothing
```

In this operation, the Client for Windows releases the memory it previously allocated for the *Item* object. Failure to release objects results in the Client for Windows eventually running out of memory. Also, the Client for Windows does not actually exit if any objects are still open.

Handling Errors

The Client for Windows throws an exception when it detects an error. In Visual Basic, exceptions can be caught with the *OnError* statement. Programs that count on exceptions to catch errors do not need to check the return value after calling a method.

A viable strategy for processing the Client for Windows errors is to execute an *On Error Resume Next* statement at program start-up and to test the value of the built-in Visual Basic *Err* variable upon return from a method. When *Err* is nonzero, an error has occurred and the *Error* object can be consulted to obtain the details (the *Error* object can be found as a property of the *Application* object). The *Error* object contains the actual error codes and the error message string.

Most methods return an error status. The type of this status is VT_I4, which in Visual Basic translates to the Long data type. The error status is either zero (successful) or nonzero (error detected). When an error has been detected, details about the problem can be obtained by consulting the *Error* object.

Property and Argument Types

The arguments and properties are listed in Chapter 7. These types can be translated into Visual Basic types and Visual C++ types by consulting the following table:

OLE Type	VisualBasic	C++	Description
VT_BSTR	String	Char Array, zero terminated	An ASCII string. Can have any type of character data, but usually holds user readable text.
VT_DISPATCH	Object	IDispatch*	A reference to an OLE object. Read the method or property to determine what type of object will be returned.
VT_VARIANT (safe array)	Array (VB 4.0 or greater only)	IVariant*	A safe array of objects. In the areas where safe arrays are used, the object type is VT_BSTR.
VT_I4	Number	long	A long integer. Can be positive or negative. The acceptable range is -2 147 483 648 to +2 147 483 647.
VT_EMPTY	(N/A)	void	No value.
VT_UNKNOWN	(N/A)	IVariant*	A structure used internally by OLE automation.
VT_BOOL	Boolean	int	A logical value with two possible values: TRUE or FALSE.

Sample Visual Basic Program

This section shows the code for a Visual Basic program that starts the Client for Windows and causes it to display the "To be indexed" workbasket. Then it displays the first item in the workbasket, whether it is a document or a folder. To keep the example readable, no error handling has been taken into account. The best way to learn from this program is to type it into Visual Basic and then trace through it by repeatedly pressing the F8 key.

```
' This example invokes the Client for Windows and causes it to display the
' To be indexed workbasket, then displays the first item in the workbasket,
' whether it is a document or a folder.
' Data declarations
  Dim VicApp As Object
  Dim Workbasket As Object
  Dim Docs As Object
  Dim Doc As Object
  Dim Item As Object
' Get the application objects
```

```

    Set VicApp = CreateObject("Vic.Application")
' Set login information
    VicApp.User = "GLEND"
    VicApp.Password = "PASSWORD"
' Log into Content Manager for iSeries
    VicApp.Logon
' Get the workbasket item
    Set Workbasket = VicApp.GetWorkbasket("To be indexed")
' Display the workbasket
    Set Docs = VicApp.Documents
    Set Doc = Docs.OpenTOC(Workbasket)
' Get next item from workbasket
    Set Item = Workbasket.NextWorkbasketItem
' Find out if the item is a folder or a document
    If (Item.Type = 1) Then
        ' Document! Display it.
        VicApp.Image.OpenDocument Item
    Else
        ' Must be a folder. Display it.
        Docs.OpenTOC Item
    End If
' Clean up
    Set Workbasket = Nothing
    Set Docs = Nothing
    Set Doc = Nothing
    Set Item = Nothing
    VicApp.Quit
    Set VicApp = Nothing

```

In this example, the Client for Windows is loaded, and then the user name and password to be used, while logging onto the default Client for Windows Library Server are configured. Next, the Client for Windows log on is executed.

After getting the "To be indexed" workbasket item, the workbasket is opened using the *Documents* object.

The next step is to get the next item in the workbasket and determine if it is a document or a folder. If it is a folder, it is passed to the *Documents* object, while a document is passed to the *Image* object.

Finally, the Client for Windows ends.

Properties and Methods of OLE Objects for Windows

This section describes the properties and methods associated with all Windows client application objects.

Application Object

The Application object gets and sets application-level states, such as log on and quit.

Properties

The Application object has the following properties.

Application

The Application property returns the Application object.

Data Type: VT_DISPATCH (Application)

Documents

The Documents property holds a collection of Document objects. A document, in Client for Windows terms, is a Table of Contents view.

Data Type: VT_DISPATCH (Documents)

Error The error information for the most recent method error.

Data Type: VT_DISPATCH (Error)

HWnd

This property returns the client's main window handle. This is a read only property.

Data Type: VT_14

Image The Image property holds the IBM Content Manager for iSeries document that is currently visible in the image viewer. If no document is visible, Image returns NULL.

Data Type: VT_DISPATCH (Image)

KeyFieldTranslation

The KeyFieldTranslation property sets the Item.KeyFields property to either translate or not translate the values that have been retrieved or set, depending on the value of the KeyFieldTranslation property.

Data Type: VT_BOOL

NewPassword

The NewPassword property is used to change the user's password. You should set this property before calling the Logon method. If the user successfully logs on, the user's password is changed. The default value is NULL.

Data Type: VT_BSTR

Password

The Password property is the password to be used when the Logon method is called to log on to the IBM Content Manager for iSeries Library Server. Reference the description of the Application object's Logon method for a description of the possible values and results.

Data Type: VT_BSTR

Server The Server property contains the name of the Library Server that is logged on to when the Logon method is called. Reference the description of the Application object's Logon method for a description of the possible values and results.

Data Type: VT_BSTR

User The User property c Application object's Logon method for a description of the possible values and results.

Data Type: VT_BSTR

Visible

The Visible property contains the visible status of the Windows Client frame window. The default value is False (0).

Data Type: VT_BOOL

Methods

The Application object supports the following methods.

Activate

This method attempts to force the client into the foreground.

Parameters: None

Returns: None

ClassArray

The ClassArray method returns a safe array of VT_BSTRs containing the names of all of the index classes defined at the time the Logon method was executed.

Parameters: None

Returns: VT_VARIANT (safe array of VT_BSTR)

ClassKeyFieldArray

The ClassKeyFieldArray method returns a safe array of VT_BSTRs containing the names of all of the key fields associated with the specified index class at the time the Logon method was executed.

Parameters: Index Class as VT_BSTR

Returns: VT_VARIANT (safe array of VT_BSTR)

ClassKeyFieldList

The ClassKeyFieldList method returns a string with all of the key fields associated with the specified index class at the time the Logon method was executed. The key fields are separated by the string separator argument.

Parameters: IndexClass as VT_BSTR, Separator as VT_BSTR

Returns: VT_BSTR

ClassList

The ClassList method returns a string with a list of all of the index classes defined at the time the Logon method was executed. The index classes are separated by the string separator argument.

Parameters: Separator as VT_BSTR

Returns: VT_BSTR

ContentClassArray

The ContentClassArray method returns a safe array of VT_BSTRs containing the names of all content classes that were defined at the time the Logon method was executed.

Parameters: None

Returns: VT_VARIANT (safe array of VT_BSTR)

ContentClassList

The ContentClassList method returns a string with all of the content classes that were defined at the time the Logon method was executed. The content classes are separated by the separator argument.

Parameters: Separator as VT_BSTR

Returns: VT_BSTR

CreateDocument

The CreateDocument method returns an *Item* object that represents a newly created document. It contains no objects (pages), and is indexed with a NOINDEX index class. The source key field is filled in with the Source argument's value, the name key field is filled in with the contents of the User property, and the timestamp key field is the exact time and date that the document was created.

Parameters: Source as VT_BSTR

Returns: VT_DISPATCH (Item)

CreateFolder

The CreateFolder method returns an Item object that represents a newly created folder. It contains no items in its TOC, and is indexed with a *NOINDEX* index class. The *Source* key field is filled in with the Source argument's value, the *UserID* key field is filled in with the contents of the User property, and the *Timestamp* key field is the exact time and date that the document was created.

Parameters: Source as VT_BSTR

Returns: VT_DISPATCH (Item)

DisableMenus

This DisableMenus method allows you to disable menu classes. You specify the menus to be disabled using the *Flags* argument. The valid values for this method are listed below:

- IP2_DISABLE_CHECKINOUT (0x001)
Prevents the user from checking items in or out
- IP2_DISABLE_DELETE (0x002)
Prevents the user from deleting items
- IP2_DISABLE_EXPORT (0x004)
Prevents the user from exporting items
- IP2_DISABLE_FAXOUT (0x008)
Prevents the user from faxing items
- IP2_DISABLE_FOLDER_FUNCTIONS (0x0010)
Prevents the user from adding items to an existing folder, adding items to a new folder, or removing items from a folder
- IP2_DISABLE_INDEX_CLASS_CHANGE (0x0020)
Prevents the user from changing to a different index class. The user can still edit the key fields for the index class.
- IP2_DISABLE_INDEX_VALUE_CHANGE (0x0040)
Prevents the user from changing to a different index class and from editing the key fields from the index class. The user can browse the menu and copy the values listed in the window. If you specify this value, the system ignores the *IP2_DISABLE_INDEX_CLASS_CHANGE* flag.
- IP2_DISABLE_NOTE_APPEND (0x0100)
Prevents the user from editing previously saved notes and from adding new notes. The user can open and copy existing notes in browse mode. When no notes exist, the Note Log window is not displayed. If you specify this value, the system ignores the *IP2_DISABLE_NOTE_EDIT* flag.
- IP2_DISABLE_NOTE_EDIT (0x0080)
Prevents the user from editing previously saved notes; however, the user can still add new notes
- IP2_DISABLE_OPTIONS (0x8000)
Prevents the user from using the **Options→Preferences** or **Options→Layout** menu options.
- IP2_DISABLE_PRINT (0x0200)
Prevents the user from printing items

- IP2_DISABLE_SEARCH (0x4000)
Prevents the user from searching
- IP2_DISABLE_WORKBASKET_ACTIVATE (0x0400)
Prevents the user from removing an item from suspended status
- IP2_DISABLE_WORKBASKET_REMOVAL (0x0800)
Prevents the user from removing items from a workbasket or routing them from one workbasket to another
- IP2_DISABLE_WORKBASKET_SUSPEND (0x1000)
Prevents the user from suspending items
- IP2_DISABLE_WORKFLOW (0x2000)
Prevents the user from starting an item in a workflow, changing an item's workflow, completing an item's workflow, or removing an item from its workflow

These values can be combined in order to disable more than one class at a time. If you call the `DisableMenus` method with a *Flags* argument of zero, the method will make the menus fully available.

You can use the optional *Hide* argument to delete the menu options instead of disabling them. However, if you delete a menu item, you cannot restore that item by setting a lower restriction value.

Parameters: *Flags* as VT_14, *Hide* as VT_VARIANT (optional, usually VT_BOOL)

Returns: VT_NONE

ExtendedPrintSetup

The `ExtendedPrintSetup` method allows the external application to modify the default printing behavior for the client. The options described are extended print features that cannot be configured from the standard user interface.

Parameters: *Comments* as VT_BOOL, *Borders* as VT_BOOL, *SinglePage* as VT_BOOL, *HorizPages* as VT_BOOL, *PageNumbers* as VT_BOOL, *NumRows* as VT_I2, *NumColumns* as VT_I2 .

Returns: VT_I4

- *Comments* is an alternate way of not printing the annotations. This option duplicates the *PrintMarkup* argument in the `Item.PrintItem` method.
- *Borders* enables or disables a single pixel line around the image. This feature is most useful if you set *SinglePage* to false (see next bullet).
- *SinglePage* can be used in conjunction with the *NumRows*, *NumColumns*, and *HorizPages* arguments to define how to arrange images on pages. If *SinglePage* is true, only one image prints on each page. If *SinglePage* is false, the other three arguments define how many images to print on each page.
- *HorizPages* is used when the *SinglePage* argument is false. *HorizPages* specifies image orientation on the printed page: true for horizontal and false for vertical.
- *PageNumbers* prints the page number on each image. If *PageNumbers* is set to true, the page number prints in the upper left corner of each image (a page might show more than one image).

- *NumRows* and *NumColumns* are used when the *SinglePage* argument is false. *NumRows* and *NumColumns* define how many images to horizontally and vertically display on a single printed page.

GetWorkbasket

The *GetWorkbasket* method returns the *Item* object associated with the workbasket specified in the *Name* argument. Note that the workbasket name is not case sensitive.

Parameters: *Name* as VT_BSTR

Returns: VT_DISPATCH (*Item*)

ItemID

The *ItemID* method returns an *Item* object with the item ID specified. Reference the *Item* object properties for a description of the *ItemID* property.

Parameters: *Item* as VT_BSTR

Returns: VT_DISPATCH (*Item*)

KeyFieldArray

The *KeyFieldArray* method returns a safe array of VT_BSTRs containing the names of all of the index classes defined at the time the *Logon* method was executed.

Parameters: None

Returns: VT_VARIANT (safe array of VT_BSTR)

KeyFieldList

The *KeyFieldList* method returns a string with all of the key fields defined at the time the *Logon* method was executed. The key fields are separated by the string separator argument.

Parameters: *Separator* as VT_BSTR

Returns: VT_BSTR

Logon The *Logon* method logs on to IBM Content Manager for iSeries. If the *User*, *Password*, and *Server* properties have all been set, a log on will be attempted with that information. If any of the previously mentioned properties were not filled in, or the initial log on attempt was unsuccessful, a log on screen will be displayed for the operator to fill in the remaining information. If the *Password* property is filled in prior to calling the *Logon* method, but the *User* property was not, the password information will be ignored.

The *Server* property is pre-initialized with the last library server that was logged onto, or "LIBSRVR2" if no successful logon has occurred.

The return value is 0 for a successful log on, or non-zero if there was an error.

Parameters: None

Returns: VT_I4

OpenBasicSearch

The *OpenBasicSearch* method displays the basic search dialog box, allowing the operator to fill in a search. Note that the resulting *Document* object is not returned.

Parameters: None.

Returns: None

OpenScan

The OpenScan method displays the scan dialog box, allowing the operator to open a scan session. Note that the resulting Document object is NOT returned.

Parameters: None.

Returns: None

OpenWorkbasket

The Workbasket method displays the Workbasket selection dialog box, allowing the user to select a workbasket to open. Note that the Document object that results is NOT returned.

Parameters: None.

Returns: None

PrintSetup

The PrintSetup method allows the external application to modify the default printing behavior for the client. Values specified with this method are saved as the default print settings, not only for OLE Automation printing, but also for user-initiated printing.

Parameters: Printer as VT_BSTR, PaperSize as VT_I2, Portrait as VT_BOOL, Copies as VT_I2, Scaling as VT_VARIANT (optional, usually VT_BOOL).

Returns: VT_I4

- *Printer* specifies the name of the printer to print to.
- *PaperSize* defines the paper type. Specify the paper type you want by assigning the number that corresponds to it (1 through 41) in the following list:
 1. Letter 8 1/2 x 11 in
 2. Letter Small 8 1/2 x 11 in
 3. Tabloid 11 x 17 in
 4. Ledger 17 x 11 in
 5. Legal 8 1/2 x 14 in
 6. Statement 5 1/2 x 8 1/2 in
 7. Executive 7 1/4 x 10 1/2 in
 8. A3 297 x 420 mm
 9. A4 210 x 297 mm
 10. A4 Small 210 x 297 mm
 11. A5 148 x 210 mm
 12. B4 (JIS) 250 x 354
 13. B5 (JIS) 182 x 257 mm
 14. Folio 8 1/2 x 13 in
 15. Quarto 215 x 275 mm
 16. 10x14 in
 17. 11x17 in
 18. Note 8 1/2 x 11 in
 19. Envelope #9 3 7/8 x 8 7/8
 20. Envelope #10 4 1/8 x 9 1/2
 21. Envelope #11 4 1/2 x 10 3/8

22. Envelope #12 4 \276 x 11
23. Envelope #14 5 x 11 1/2
24. C size sheet
25. D size sheet
26. E size sheet
27. Envelope DL 110 x 220mm
28. Envelope C5 162 x 229 mm
29. Envelope C3 324 x 458 mm
30. Envelope C4 229 x 324 mm
31. Envelope C6 114 x 162 mm
32. Envelope C65 114 x 229 mm
33. Envelope B4 250 x 353 mm
34. Envelope B5 176 x 250 mm
35. Envelope B6 176 x 125 mm
36. Envelope 110 x 230 mm
37. Envelope Monarch 3.875 x 7.5 in
38. 6 3/4 Envelope 3 5/8 x 6 1/2 in
39. US Std Fanfold 14 7/8 x 11 in
40. German Std Fanfold 8 1/2 x 12 in
41. German Legal Fanfold 8 1/2 x 13 in

- *Portrait* defines the print orientation (true = Portrait, false = Landscape).
- *Copies* specifies the number of copies to print.
- *Scaling* specifies whether the printing occurs as "fit to page" size or "normal" size. If *Scaling* is set to True (non zero) or omitted, printing is done as "fit to page". If *Scaling* is set to False, printing is done as "normal" size.

QueryPrivilege

The QueryPrivilege method allows an external application to determine the actual privileges for a user who is currently logged on. The application can check general privileges or specific privileges (such as those for an index class or workbasket).

Parameters: Authority as VT_I4, Context as VT_VARIANT (optional, VT_BOOL(Item) or VT_BSTR).

Returns: VT_BOOL

- *Authority* defines which privilege to check. You can set this value to any of the OIM_ values supported by the Folder Manager function **Ip2QueryPrivBuffer**.
- *Context* determines evaluated privileges for different contexts. If you do not enter a value for *Context*, the user's general privilege is returned. You can also set *Context* to one of the following:
 - A dispatch to an Item object: a folder, document, or workbasket
 - The name of an index class (VT_BSTR)
 - The name of a workflow (VT_BSTR)

Quit The Quit method ends the Client for Windows application. All open documents (TOCs), any image viewer sessions, and all outstanding Item and Items objects are closed.

Parameters: None

Returns: None

Search

The Search method returns an Item that represents the results of a search conducted on the system fileroot with an optional index class and key field wildcard search string. The search results folder is deleted automatically when it is closed, unless the index class is changed. The format of the search string is defined in “LIBSEARCHCRITERIASTRUCT (Search Criteria Information Structure)” on page 147.

When TypeFilter=1, only folders are returned.

When TypeFilter=2, only documents are returned.

Any other TypeFilter value returns both documents and folders.

If WipFilter=1, items not in a workflow are returned.

If WipFilter=2, items that are currently in a workflow are returned.

If WipFilter=4, items that were cancelled from a workflow are returned.

If WipFilter=8, items that completed a workflow are returned.

The WipFilter values may be combined with a binary OR operator.

If SuspendFilter=1, active items are returned. active or suspended items.

If SuspendFilter=2, suspended items are returned.

Any other SuspendFilter value returns items that are either suspended or activated.

Parameters:

IndexClass as VT_BSTR (optional)

SearchString as VT_BSTR (optional)

TypeFilter as VT_VARIANT (optional, usually VT_I2)

WipFilter as VT_VARIANT (optional, usually VT_I2)

SuspendFilter as VT_VARIANT (optional, usually VT_I2)

Returns: VT_DISPATCH (Item)

SetPrintRect

The SetPrintRect method allows you to define a rectangle that contains the images when they are printed on the page. Values specified with this method are saved as the default print settings, not only for OLE Automation printing, but also for user-initiated printing.

Parameters: RectLeft as VT_I2, RectTop as VT_I2, RectRight as VT_I2, RectBottom as VT_I2.

Returns: None

The four arguments define the distance in millimeters of each box side from the upper left hand corner of the paper. You can reset the print rectangle to “none” by calling the SetPrintRect method again and setting all arguments set to 0.

Attention: If you specify a rectangle that doesn’t fit on the paper, some or all of the image does not appear on your print out.

WorkbasketArray

The WorkbasketArray method returns a safe array of VT_BSTRs containing the names of all the workbaskets defined at the time the Logon method was executed.

Parameters: None

Returns: VT_VARIANT

WorkbasketList

The WorkbasketList method returns a string with a list of all of the workbaskets defined at the time the Logon method was executed. The workbaskets are separated by the string separator argument.

Parameters: Separator as VT_BSTR

Returns: VT_BSTR

Document Object

The *Document* object holds information about a Table of Contents (TOC).

Properties

Application

The Application property returns the Application object.

Data Type: VT_DISPATCH (Application)

Count The Count property returns the number of items that are listed in the TOC.

Data Type: VT_I4

Item The Item property returns the Item object that is associated with this Document (TOC).

Data Type: VT_DISPATCH (Item)

Page The Page property contains the selected page number. This property is valid only for documents, not workbaskets or folders. The default value is 0.

Data Type: VT_I4

PageCount

The PageCount property contains the number of pages in a document. This property is valid only for documents, not workbaskets or folders. The default value is 0. This is a read only property.

Data Type: VT_I4

Parent The Parent property returns the parent of the Document object (which is the Documents collection object).

Data Type: VT_DISPATCH (Documents)

SelectedCount

The SelectedCount property returns the number of items that are selected in the TOC.

Data Type: VT_I4

Type The Type property returns the type of item that is open in the document: a folder, workbasket, or a document. The actual values are:

1 - Document

2 - Folder

3 - Workbasket

1024 - Scan (the basic scan viewer, no other property or method works on this type)

The default value is 0 (error). This is a read only property.

Data Type: VT_I4

Methods

The Document object supports the following methods.

Activate

The Activate method brings the TOC window associated with this document to the foreground.

Parameters: None

Returns: VT_I4

CaretIndex

The CaretIndex method returns the index of the caret item (the item that contains the dotted-line rectangle in the grid) in a folder or workbasket.

Parameters: None

Returns: VT_I4

ClearSelect

The ClearSelect method clears all of the current selections in the TOC.

Parameters: None

Returns: VT_I4

Close

The Close method closes the window associated with the associated document (TOC) and removes the document from the *Documents* collection. The remaining Document objects in the collection will be shifted down to prevent gaps in the collection.

Parameters: VT_VARIANT (optional, usually VT_BOOL)

Returns: VT_I4

CloseIt

The CloseIt method is the same as the Close method. It is implemented solely to support VisualBasic, which uses Close as a reserved word. The CloseIt method closes the window associated with the associated document (TOC) and removes the document from the Documents collection. The remaining Document objects in the collection will be shifted down to prevent gaps in the collection.

Parameters: VT_VARIANT (optional, usually VT_BOOL)

Returns: VT_I4

DisplayPage

The DisplayPage method forces the page specified to be displayed in a document. This method is valid only for documents, not workbaskets or folders.

Parameters: Page as VT_I4

Returns: VT_I4

FirstPage

The FirstPage method displays the first page in a document. This method is valid only for documents, not workbaskets or folders.

Parameters: None

Returns: VT_I4

IndexedItem

The IndexedItem method returns a single item from Document based on its index (specified with the Index argument) from a folder or workbasket.

Parameters: Index as VT_I4

Returns: VT_DISPATCH (Item)

LastPage

Displays the last page in a document. This method is valid only for documents, not workbaskets or folders.

Parameters: None

Returns: VT_I4

Maximize

The Maximize method maximizes the Document object in the main client window, hiding all other Document objects.

Parameters: None

Returns: VT_I4

Minimize

The Minimize method minimizes the Document object in the main client window.

Parameters: None

Returns: VT_I4

NextPage

The NextPage method displays the next page (current page, plus 1) in a document. This method is valid only for documents, not workbaskets or folders.

Parameters: None

Returns: VT_I4

PreviousPage

The PreviousPage method displays the previous page (current page, minus 1) in a document. This method is valid only for documents, not workbaskets or folders.

Parameters: None

Returns: VT_I4

Restore

The Restore method restores the Document object in the main client window to its original state (neither minimized or maximized).

Parameters: None

Returns: VT_I4

Selections

The Selections method returns an Items collection containing all of the *Item* objects that are selected in the Document (TOC).

Parameters: None

Returns: VT_DISPATCH (Items)

SelectRange

The SelectRange method selects a range of items in the TOC. The arguments are the zero-based index of the first and last items to be selected.

Parameters:

First as VT_I4

Last as VT_I4

Returns: VT_I4

Zoom The Zoom method changes the zoom ration of the Document object. For example, if you set the zoom ratio to 100, the image is shown at full size, pixel for pixel. If you set the zoom ration to 50, the image is shown in half height. Zoom only works on documents, not folders or workbaskets.

Parameters: Percent as VT_I4

Returns: VT_I4

ZoomFit

The ZoomFit method allows you to fit the document image into the viewing rectangle. The Type argument specifies how to fit: 1 means fit height, 0 means fit width. ZoomFit only works on documents, not folders or workbaskets.

Parameters: Fit as VT_I4

Returns: VT_I4

ZoomRect

ZoomRect allows you to specify a rectangle to zoom to in the Document object. The left, top, right, and bottom arguments specify the bounding rectangle to display as large as possible in the viewing rectangle (the viewer window). The arguments are specified in pixels. ZoomRect only works on documents, not folders or workbaskets.

Parameters:

Left as VT_I4

Top as VT_I4

Right as VT_I4

Bottom as VT_I4

Returns: VT_I4

Documents Object

The Documents collection object is a collection of all of the open Document objects (TOCs).

Properties

The Document object has the following properties.

Active

The Active property holds the index of the Document object that currently has the focus. This is a read only property.

Data Type: VT_I4

Application

The Application property returns the Application object.

Data Type: VT_DISPATCH (Application)

Count The Count property holds the number of Document objects currently in the collection.

Data Type: VT_I4

Parent The Parent property returns the parent of the Documents collection object (which is the Application object).

Data Type: VT_DISPATCH (Application)

Methods

The Document object supports the following methods.

Cascade

The Cascade method arranges all of the open Document objects that are not minimized in a cascaded manner.

Parameters: None.

Returns: VT_I4

Close The Close method closes all windows associated with the Documents objects and removes the Document objects from the Documents collection.

Parameters: None

Returns: None

CloseIt

Attention: The CloseIt method is the same as the Close method. It is implemented solely to support VisualBasic, which uses Close as a reserved word. The Close method closes all windows associated with the Documents objects and removes the Document objects from the Documents collection.

Parameters: None

Returns: VT_I4

Item The Item method returns one of the Document objects contained in the collection.

Parameters: Index as VT_I4

Returns: VT_DISPATCH (Document)

OpenDocument

The OpenDocument method creates a new Document object for the document and adds it to the Documents collection. If the Browse argument is set to TRUE, the document is opened without being locked, allowing other users to open it.

Parameters:

Index as VT_DISPATCH (Item)

Browse as VT_VARIANT (optional, usually VT_BOOL)

Returns: VT_DISPATCH (Document)

OpenTOC

The OpenTOC method creates a new Document object for the specified workbasket or folder and adds it to the Documents collection. If the Browse argument is set to TRUE, the folder is opened without being locked, allowing other users to open it. Browse has no affect on workbaskets.

Parameters:

Index as VT_DISPATCH (Item)

Browse as VT_VARIANT (optional, usually VT_BOOL)

Returns: VT_DISPATCH (Document)

Tile The Tile method arranges all of the open Document objects that are not minimized in a tiled manner. The Vertical argument specifies if the objects should be set primarily vertically (non-zero) or horizontally (zero).

Parameters: Vertical as VT_I4

Returns: VT_I4

Error Object

The Error object describes the details about any error that may have happened while executing a method in Client for Windows.

Properties

The Error object has the following properties.

ErrorMessage

The ErrorMessage property contains a descriptive error code describing what went wrong and what Client for Windows was doing at the time.

Data Type: VT_BSTR

ExtReturnCode

The ExtReturnCode property contains the extended return code that was returned when the error was detected.

Data Type: VT_14

ReturnCode

When detecting an error, the ReturnCode property contains the error code. OLE Automotation methods now return standardized error codes—either uniform four digit codes described in the "Messages and Codes" manual or values described in the frnwole.h header file, as shown in Table 3.

Data Type: VT_14

Table 3. Standardized OLE API Return Codes

OLEAPI_RC_NOT_LOGGED_ON	12000
OLEAPI_RC_INVALID_INDEXCLASS	12001
OLEAPI_RC_INSUFFICIENT_MEMORY	12002
OLEAPI_RC_NO_ITEMS_FOUND	12003
OLEAPI_RC_INVALID_WORKBASKET	12004
OLEAPI_RC_ALREADY_LOGGED_ON	12005
OLEAPI_RC_INVALID_ARGUMENT	12006

Table 3. Standardized OLE API Return Codes (continued)

OLEAPI_RC_NO_DOC_OPEN	12007
OLEAPI_RC_INVALID_ITEM	12008
OLEAPI_RC_INDEX_OUT_OF_RANGE	12009
OLEAPI_RC_INVALID_KEYFIELD	12010
OLEAPI_RC_ERROR_PRINTING	12011
OLEAPI_RC_INVALID_CONTENT_CLASS	12012
OLEAPI_RC_ITEM_NOT_FOLDER	12013
OLEAPI_RC_ITEM_NOT_WORKBASKET	12014
OLEAPI_RC_ITEM_NOT_WORKFLOW	12015
OLEAPI_RC_ERROR_GETTING_PART	12016
OLEAPI_RC_ERROR_UNLOCKING	12017
OLEAPI_RC_INVALID_DOCUMENT	12018
OLEAPI_RC_NOT_TOC_DOCUMENT	12019
OLEAPI_RC_INSUFFICIENT_PRIVS	12020
OLEAPI_RC_NO_SELECTIONS	12021
OLEAPI_RC_NOT_DOC_DOCUMENT	12022
OLEAPI_RC_ITEM_NOT_TOC	12023
OLEAPI_RC_ITEM_NOT_DOCUMENT	12024
OLEAPI_RC_TEMP_FOLDER	12030
OLEAPI_RC_VALIDATION_ERROR	12040
OLEAPI_RC_UNABLE_TO_QUIT	12100
OLEAPI_RC_FAX_NOT_INSTALLED	12110
OLEAPI_RC_FAX_GEN_ERROR	12111
OLEAPI_RC_FAX_EMPTY_TOC	12112
OLEAPI_RC_FAX_NODOCSIN_TOC	12113

Methods

The Error object does not have any methods.

Image Object

Attention: In place of the Image Object, we recommend using the Document and Documents Objects to permit the ability to open more than one document at a time.

The Image object holds the currently visible document.

Properties

The *Image* object supports the following properties.

Application

The Application property returns the Application object.

Data Type: VT_DISPATCH (Application)

Item The Item property returns the *Item* object that is associated with this Image.

Data Type: VT_DISPATCH (Item)

Page The Page property contains the selected page number.

Data Type: VT_I4

Parent The Parent property returns the parent of the Image object (which is the Application object).

Data Type: VT_DISPATCH (Application)

Methods

The Image object supports the following methods.

Close The Close method closes all windows associated with the Image object. If the Save argument is True, any changes to the object are saved. If the Save argument is False, changes are thrown away. If the Save argument is not specified, a message box asks the user if they want to save the changes or not.

Parameters: Save as VT_VARIANT (optional, usually VT_BOOL)

Returns: None

CloseIt

Attention: The CloseIt method is the same as the Close method. It is implemented solely to support VisualBasic which uses Close as a reserved word. The CloseIt method closes all windows associated with the Image object. If the Save argument is True, any changes to the object are saved. If the Save argument is False, changes are thrown away. If the Save argument is not specified, a message box asks the user if they want to save the changes or not.

Parameters: Save as VT_VARIANT (optional, usually VT_BOOL)

Returns: None

DisplayPage

The DisplayPage method forces the page specified to be displayed in the image viewer.

Parameters: Page as VT_I4

Returns: VT_I4

FirstPage

The FirstPage method displays the first page in the viewer.

Parameters: None

Returns: VT_I4

LastPage

The LastPage method displays the last page in the viewer.

Parameters: None

Returns: VT_I4

NextPage

The NextPage method displays the next page (current page + 1) in the viewer.

Parameters: None

Returns: VT_I4

OpenDocument

The OpenDocument method opens a new IBM Content Manager for iSeries

document in the image viewer. The argument Index is the item that is to be opened. An Item error will occur if the item is not a workbasket or folder.

Parameters: Index as VT_DISPATCH (Item)

Returns: VT_I4

PreviousPage

The PreviousPage method displays the previous page (current page – 1) in the viewer.

Parameters: None

Returns: VT_I4

Item Object

The Item object represents an item like a folder, workbasket, or document.

Properties

The Item object supports the following properties.

Application

The Application property returns the Application object.

Data Type: VT_DISPATCH (Application)

CheckedStatus

The CheckedStatus property returns the user who has the item checked out, if any.

Data Type: VT_BSTR

Class The Class property is the index class of the item. Changes to the key field values are not updated until you call the UpdateIndex method. This is a read/write property.

Data Type: VT_BSTR

ItemID

The ItemID property is a string that uniquely defines each item in the IBM Content Manager for iSeries fileroom.

Data Type: VT_BSTR

Name The Name property returns IBM Content Manager for iSeries's name for the item. This property is based on the key field selected as the identifier (if any) when the index class was created. If the item is a workbasket the workbasket name is returned.

Data Type: VT_BSTR

PartCount

The PartCount property returns the number of parts stored in a document.

Data Type: VT_I4

Parent The Parent property returns the parent of the Image object (which can be the Application object or an Items object).

Data Type: VT_DISPATCH (Application or Items)

Priority

The Priority property returns the workbasket priority of the item. Valid

values are 1 to 31,999, where 1 is the lowest priority. If the item is not in a workbasket, the Priority property returns the class default priority, and is read-only.

Data Type: VT_14

SystemAssigned

Returns TRUE if the workbasket is a system assigned workbasket.

Data Type: VT_BOOL

TOCCount

The TOCCount property returns the number of items that are indexed in this table of contents.

Data Type: VT_14

Type The Type property returns the item type of the item. A value of 1 means a document, 2 means folder, and 3 means workbasket.

Data Type: VT_14

Methods

The Item object supports the following methods.

Activate

The Activate method removes the suspended status from a suspended item.

Parameters: None

Returns: VT_I4

AddAnnotationPart

The AddAnnotationPart method can be used to add an annotation part to an existing document. The Path argument must be a full path to the new annotation part to be used with the document. If an annotation part already exists, it will be replaced by the new annotation file. Note that an extension of ".T_L" is assumed and will be used even if a different extension is provided.

Parameters: Path as VT_BSTR

Returns: VT_I4

AddPart

The AddPart method adds a file as an object to the item. You must specify a full path and a content class. Optionally, you can specify that the Library Server should choose the Object Server and Collection for a new part according to its rules (usually the User's default Object Server and Collection).

Parameters:

Path as VT_BSTR

ContentClass as VT_BSTR

SMSOption as VT_VARIANT (optional, usually VT_BOOL)

If SMSOption is set to TRUE (non-zero) or omitted, the Index Class's default Object Server and collection will be used (original behavior). If SMSOption is set to FALSE (0), the Library Server will choose the Object Server and Collection, based on the configuration (usually the defaults for the user).

Returns: VT_14

AddToFolder

The AddToFolder method adds the Item to the folder specified as another Item object.

Parameters: Folder as VT_DISPATCH (Item)

Returns: VT_I4

ChangeNotes

The ChangeNotes method saves the argument value as the note log.

Parameters: Value as VT_BSTR

Returns: VT_I4

ChangeWorkflow

The ChangeWorkflow method allows you specify a new workflow to send the item through. The new workflow is specified by name with the WorkFlow argument.

Parameters: WorkFlow as VT_BSTR

Returns: VT_I4

CheckIn

The CheckIn method checks the item in, allowing anyone to modify it.

Parameters: None

Returns: VT_I4

CheckOut

The CheckOut method checks the item out to the current user, disabling anyone else from modifying it.

Parameters: None

Returns: VT_I4

Close

The Close method unlocks the item previously locked with the Open method or NextWorkbasketItem (the resulting item, not the workbasket).

Parameters: None

Returns: VT_I4

CloseIt

Attention: The CloseIt method is the same as the Close method. It is implemented solely to support VisualBasic, which uses Close as a reserved word. The CloseIt method unlocks the item previously locked with the Open method or NextWorkbasketItem (the resulting item, not the workbasket).

Parameters: None

Returns: VT_I4

CloseNotes

The CloseNotes method closes the open note log without saving any changes.

Parameters: None

Returns: VT_I4

CloseParts

The CloseParts method closes all of the open part files (pages) without saving any changes.

Parameters: None

Returns: VT_I4

CompleteWorkflow

The CompleteWorkflow method marks the item as successfully finishing a workflow.

Parameters: None

Returns: VT_I4

Delete The Delete method removes the item from the fileroom. This is a non-recoverable operation, so use this method with care.

Parameters: None

Returns: VT_I4

DeletePart

The DeletePart method deletes the specified object (part) from the item.

Parameters: Index as VT_I4

Returns: VT_I4

FaxItem

The FaxItem method sends the item to the fax subsystem if it is loaded. The argument withSubFolderContents, if specified and set to True (non-zero), enables you to fax the documents contained in folders.

Parameters: withSubFolderContents as VT_VARIANT (optional, usually VT_BOOL)

Returns: VT_I4

GetAnnotationFile

The GetAnnotationFile method retrieves the annotation file for the item.

Parameters: None

Returns: VT_BSTR

GetHistoryLog

The GetHistoryLog method retrieves the work history for the item.

Parameters: None

Returns: VT_BSTR

GetKeyFields

The GetKeyFields method returns the value for the given key field of an item.

Parameters: Name as VT_BSTR

Returns: VT_BSTR

GetNotes

The GetNotes method retrieves the text of the note log from IBM Content Manager for iSeries and returns it to the calling application. The item is checked out in IBM Content Manager for iSeries the first time you call this method.

Parameters: None

Returns: VT_BSTR

GetPartContentClass

The GetPartContentClass method returns the content class of the part file specified with the Index argument.

Parameters: Index as VT_I4

Returns: VT_BSTR

GetPartFile

The GetPartFile method retrieves an object file from IBM Content Manager for iSeries, stores it on the local workstation, and returns the full path to the temporary file. The Item is checked out in IBM Content Manager for iSeries the first time you call this method.

Parameters: Index as VT_I4

Returns: VT_BSTR

GetParentFolders

The GetParentFolders method returns an Items collection of folders. Each of these folders contains the document of the folder that calls the method.

Parameters: None

Returns: VT_DISPATCH (Items)

GetTOCItem

The GetTOCItem method returns the Item object specified from the TOC.

Parameters: Index as VT_I4

Returns: VT_DISPATCH (Item)

NextWorkbasketItem

The NextWorkbasketItem method returns the next available item by order of priority in a workbasket.

Parameters: None

Returns: VT_DISPATCH (Item).

Open The Open method locks the item. No other user can modify index information or modify parts when the item is locked. You must use the Close or CloseIt methods to unlock the item.

Parameters: None

Returns: None

PreStage

The PreStage method stages an off-line part for future retrieval. Call this method if Item.GetPartFile returns a 6265 (SIM_RC_OBJECT_BEINGPROMOTED) exception, which indicates that the part object is on an off-line storage device.

Parameters: Index as VT_I4

Returns: None

PrintItem

The PrintItem method prints the item to the currently selected printer using the current print options. If ShowDialog is true, the print dialog displays where the user can select a different printer, modify options, or cancel printing.

Parameters: ShowDialog as VT_BOOL, PrintImage as VT_VARIANT, StartPage as VT_VARIANT, EndPage as VT_VARIANT, PrintMarkup as

VT_VARIANT, PrintIndex as VT_VARIANT, PrintNoteLog as VT_VARIANT, PrintTOC as VT_VARIANT, PrintContents as VT_VARIANT

Returns: VT_I4

- *PrintImage* (optional) specifies whether or not to print the base parts of the document, also known as images. If you also select *PrintMarkup* (optional), any defined annotations are printed on the image.
- *StartPage* (optional) and *EndPage* (optional) specify the desired base part page ranges to print. The pages are numbered starting from 1.

Examples:

- To print the middle three pages of a five-page document, set *StartPage* to 2 and *EndPage* to 4.
- To print an entire document, set *StartPage* to 1 and *EndPage* to 10000 or some other sufficiently-large number.
- *PrintIndex* and *PrintNotelog* allow you to specify whether the indexing and note log information prints for documents and folders. Workbaskets ignore these arguments. However, you can print out the note logs and index information for the documents and folders contained in the workbasket by setting *PrintIndex* and *PrintNotelog* to true in workbaskets.
- *PrintTOC* and *PrintContents* specify how to print workbaskets and folders. If *PrintTOC* is true, the list of items contained in the folder or workbasket prints. If *PrintContents* is true, the folders and documents contained in the table of contents prints as well.

RefreshTOC

The RefreshTOC method re-samples the TOC of a workbasket or folder. If you did not call this method any changes to a workbasket or folder's TOC will not be recognized by methods in the Item class.

Parameters: None

Returns: VT_I4

RemoveFromFolder

The RemoveFromFolder method removes the item from the folder specified as an argument.

Parameters: Folder as VT_DISPATCH (Item)

Returns: VT_I4

RemoveFromWorkbasket

The RemoveFromWorkbasket method removes the item from the workbasket specified as an argument.

Parameters: Workbasket as VT_DISPATCH (Item)

Returns: VT_I4

RemoveFromWorkflow

The RemoveFromWorkflow method marks the item as being canceled from workflow.

Parameters: None

Returns: VT_I4

RouteToWorkbasket

The RouteToWorkbasket method adds this item to a workbasket, removing it from any workbasket it is currently in. The workbasket is specified by its

Item object. If Force is specified as TRUE, the item is added to the workbasket, even if the workbasket is already full.

Parameters:

Workbasket as VT_DISPATCH (Item)

Priority as VT_VARIANT (optional, usually VT_I4)

Force as VT_VARIANT (optional, usually VT_BOOL)

Returns: VT_I4

SavePart

The SavePart method saves any changes that occurred to the part file specified and its annotation file.

Returns: VT_I4

SetKeyFields

The SetKeyFields method sets the value for the given field of an item. To store updated key fields to the server, you must call the UpdateIndex method.

Parameters: Name as VT_BSTR; NewValue as VT_BSTR

Returns: None

StartWorkflow

The StartWorkflow method adds the item into the specified workflow.

Parameters:

Workflow as VT_BSTR

Workbasket as VT_VARIANT (optional, usually VT_DISPATCH)

Priority as VT_VARIANT (optional, usually VT_I4)

Returns: VT_I4

Suspend

The Suspend method causes the item to be suspended, pending some future event. This event is a time and date, but could also be an item being included in a folder item.

Parameters:

Timestamp as VT_VARIANT (optional, usually VT_BSTR)

TimeoutWorkbasket as VT_VARIANT (optional, usually VT_DISPATCH)

Classes as VT_VARIANT (optional, usually VT_BSTR)

Criteria as VT_VARIANT (optional, usually VT_I4)

ReadyWorkbasket as VT_VARIANT (optional, usually VT_DISPATCH)

If *Timestamp* is specified, the item is suspended, pending a time event. When the time event is triggered, the item is activated and placed in the *TimeOutWorkbasket* workbasket. The *Timestamp* argument must be in a format like the following example:

```
1997-09-30-08.05.23.000000
```

If *Classes* is specified (only valid for folder items), the item is suspended, pending a time event or a folder event. When the time event is triggered, the item is activated and placed in the *TimeOutWorkbasket* workbasket. If the folder event is triggered before the timeout, the item is activated and placed in the *ReadyWorkbasket* workbasket.

The optional *Classes* argument is a string containing a list of index classes separated by semicolons (;). This list is used to indicate which index classes will trigger an activation.

The optional *Criteria* argument, which is only valid for folder items, should be zero (0) to indicate an OR condition, or one (1) to indicate an AND condition. This condition is used when determining if one or all of the index classes specified in the *Classes* argument must be indexed before the folder is activated.

Returns: VT_I4

UpdateIndex

The UpdateIndex method saves any changes that you have made to the Index Class and/or key fields (using the Class property and/or the SetKeyFields method). Until this method is called no changes are stored.

Parameters: None

Returns: VT_I4

Items Collection

The Items collection holds a list of Item objects, allowing you to access the contained objects. An Items collection typically is a result of the Document method SelectionList.

Properties

Application

The Application property returns the Application object.

Data Type: Application

Count The Count property returns the number of Item objects referenced in the Items collection.

Data Type: VT_I4

Parent The Parent property returns the parent of the Items collection (which is usually a Document object).

Data Type: VT_DISPATCH (Document)

Methods

_NewEnum

The _NewEnum method returns an unknown which supports the IID_IEnumVARIANT. _NewEnum is a restricted method that cannot be invoked like the other methods. It is used to implement loop constructs in macro languages such as Visual Basic.

Parameters: None

Returns: VT_UNKNOWN

Close The Close method closes the *Items* collection.

Parameters: None.

Returns: VT_I4

CloseIt

Attention: The CloseIt method is the same as the Close method. It is

implemented solely to support VisualBasic which uses Close as a reserved word. The CloseIt method closes the Items collection.

Parameters: None

Returns: VT_I4

Item The Item method returns an Item object from the Items collection.

Parameters: Index as VT_I4

Returns: VT_DISPATCH (Item)

Chapter 6. Sample High-Level Programming Interface

Sample High-Level Programming Interface for Visual Basic

The Content Manager for iSeries client high-level programming interface is a set of frequently used folder and document management functions. These high-level functions have a simple call interface reflecting how users access documents and folders in Content Manager for iSeries. Some highlights of the Content Manager for iSeries client high-level programming interface using Visual Basic are as follows:

- Approximately 30 functions for frequently used folder and document management functions
- Single workstation logon to Content Manager for iSeries by means of the Client for Windows application
- Visual Basic OLE automation source code provided

In addition, the Client for Windows can allow multiple applications to access Content Manager for iSeries simultaneously.

General Use

The Content Manager for iSeries client high-level programming interface interacts with the basic components of the Content Manager for iSeries data model: documents, folders, and workbaskets. A Content Manager for iSeries document consists of a set of closely related objects or parts.

The Content Manager for iSeries client high-level programming interface provides functions to create, view, update and delete typical Content Manager for iSeries documents composed of a *single base part* (for example a scanned document or word processing file) and a *single note part*. Use of the Content Manager for iSeries high-level programming interface with documents containing multiple base parts can produce unexpected or undesired results. For additional information about the Content Manager for iSeries data model, see "Understanding the Logical Data Model" on page 5.

The Client for Windows ' OLE automation interface does provide the ability to manipulate multiple base part documents. Because Visual Basic source code is provided, the user might want to customize the VHLPI to handle other document compositions.

Lists of data returned by VHLPI functions can be filtered based upon the privileges set for the user ID that has logged on. In addition, the user should be aware that index class and attribute names specified as parameters to VHLPI functions are normally case-sensitive.

Visual Basic Parameters and Variables

All Visual Basic variables passed to VHLPI functions as parameters should be of type Variant or Variant Array. If a Variant Array is passed, the size of the array, excluding element index 0, should be contained in element 0 of the array.

NULL values can be set by assigning the variable to an empty string, "".

There are several global variables which are included with the VHLPI code module, FRNWWFVB.BAS. These global variables can be accessed by any Visual Basic program which includes FRNWWFVB.BAS. The global variables are as follows:

- **Vh1App1Obj** - Client for Windows ' Application Object
- **Vh1DocsObj** - Client for Windows ' Documents Collection Object
- **Vh1ErrorObj** - Client for Windows ' Error Object

These global variables are created via the **VbVhlLoadFuncs** function and they are freed by the **VbVhlDropFuncs** function. A Visual Basic program must call **VbVhlLoadFuncs** before using VHLPI functions, and should call **VbVhlDropFuncs** before ending to free these objects.

Once these variables have been created, the Visual Basic program can invoke methods or get/set properties associated with them. For instance, to find out what server the Client for Windows is logged on to, the following could be executed:

```
' Create Objects
u1RC = VbVhlLoadFuncs

' Get what server is logged on
Server$ = Vh1App1Obj.Server

' Display the server name
MsgBox "The server is " & Server$
```

Access to the Client for Windows

The Client for Windows can be used to maintain a constant logon session with Content Manager for iSeries. When started, this program logs on to Content Manager for iSeries and then waits for operator commands. Once logged on, other applications through the OLE automation interface can use the Content Manager for iSeries logon session established.

By using the Client for Windows logon session, other applications do not need to logon to Content Manager for iSeries, instead they must create an OLE automation Application Object from the Client for Windows . This can be done by executing the following:

```
Set Vh1App1Obj = CreateObject("Vic.Application")
```

where Vh1App1Obj is the global variable object included in the VHLPI code module, FRNWWFVB.BAS.

The **VbVhlLoadFuncs** function does this processing, plus initializes other global data objects. It is *recommended* that Visual Basic programs use the **VbVhlLoadFuncs** and **VbVhlDropFuncs** to get and end access to the Client for Windows .

The above description pertains to the situation where the Client for Windows is started and logged on before subsequent Visual Basic applications are executed. If this is not the case, it will be necessary for the Visual Basic application to issue logon and logoff commands as discussed in the next section.

Using Logon/Logoff with the Client for Windows

If the Client for Windows is not started and logged on before the Visual Basic application is executed, the application must call **VbVhlLogon** instead of **VbVhlLoadFuncs**. **VbVhlLogon** will cause the Client for Windows to be started and then issue the *Logon* method to logon to Content Manager for iSeries.

Once the Client for Windows is logged on to Content Manager for iSeries, any subsequent attempt to logon, even if the user ID or server information is different, does *not* cause another logon attempt. All subsequent logons will simply use the original logon session and no error indication will be provided.

The **VbVhLogoff** will issue the *Logoff* method and close the Client for Windows , even if other applications are using the logon session. If it is not desired to terminate the Client for Windows , then **VbVhDropFuncs** should be used to terminate access only for the current application.

Samples of High Level Programming Interface APIs for Windows

VbVhAddFolderItem (Add an Item to a Folder)

Format

```
VbVhAddFolderItem( ItemId, FolderId )
```

Purpose

Use this function to add a document or folder (specified by its Item Id) to an existing folder (specified by the folder's Item Id).

Parameters

ItemId — input

The Item Id of the document or folder which is to be added to the folder.

FolderId

— input

The Item Id of the folder.

Guidelines for Use

The Item Ids for both the item to add and the folder must be valid.

Visual Basic Source Code

```
Function VbVhAddFolderItem (ItemID, FolderId)
```

```
    ' Declarations
    Dim ItemObj As Object
    Dim FolderObj As Object

    ' Setup Error handler
    On Error GoTo Vh1AddFolderError
    u1RC = 0

    ' Get the Folder Object
    Set FolderObj = Vh1App1Obj.ItemID(FolderId)

    ' Get the ItemID Object
    Set ItemObj = Vh1App1Obj.ItemID(ItemID)

    ' Put ItemId into Folder
    u1RC = ItemObj.AddToFolder(FolderObj)
Vh1AddFolderEnd:

    ' Free the objects
    Set ItemObj = Nothing
    Set FolderObj = Nothing
```

VbVhlAddFolderItem

```
' Set return value to error code
VbVhlAddFolderItem = u1RC

Exit Function

VhlAddFolderError:

' Set return value to error code
u1RC = VhlErrorObj.ReturnCode

Resume VhlAddFolderEnd

End Function
```

VbVhlAdminItemNoteLog (Administer Document Note Logs)

Format

VbVhlAdminItemNoteLog(*ItemID*, *FuncInd*, *NoteText*)

Purpose

Use this function to replace, delete, get or append notes to an item's note log.

Parameters

ItemID — input

The Id of the Item.

FuncInd

— input

The function indicator which must be one of the following —

"APPEND"

Append *NoteText* to the item's note log.

"DELETE"

Delete the item's note log.

"REPLACE"

Replace the item's note log with *NoteText*.

"GET" Copy item's note log text to *NoteText*.

NoteText

— input/output

The Visual Basic variable name containing the text value of the Note.

- If *FuncInd* = *GET*, then the function copies the item's note log text into this Visual Basic variable.
- If *FuncInd* = *REPLACE* the function replaces the requested item's note log with the contents of this Visual Basic variable.
- If *FuncInd* = *APPEND* the function appends the text contained in this Visual Basic variable to the requested item's note log.

Guidelines for Use

The Item Id for the document must be valid.

Visual Basic Source Code

```

Function VbVhlAdminItemNoteLog (ItemId, FuncInd, NoteText)

    ' Declarations
    Dim ItemObj As Object

    ' Setup Error handler
    On Error GoTo VhlAdminNoteError
    u1RC = 0

    ' Get the Item object
    Set ItemObj = VhlApplObj.ItemID(ItemId)

    ' Determine what to do
    Select Case FuncInd
    Case "APPEND"
        OldNoteText = ItemObj.GetNotes
        u1RC = ItemObj.ChangeNotes(OldNoteText & NoteText)
    Case "DELETE"
        u1RC = ItemObj.ChangeNotes("")
    Case "REPLACE"
        u1RC = ItemObj.ChangeNotes(NoteText)
    Case "GET"
        NoteText = ItemObj.GetNotes
    End Select

VhlAdminNoteEnd:

    ' Free the object
    Set ItemObj = Nothing

    ' Set return value to error code
    VbVhlAdminItemNoteLog = u1RC

    Exit Function

VhlAdminNoteError:

    ' Set return code to error code
    u1RC = VhlErrorObj.ReturnCode

    Resume VhlAdminNoteEnd

End Function

```

VbVhlChangeItemIndex (Change an Item's Index Class)

Format

```
VbVhlChangeItemIndex( ItemId, ClassName, AttrName(), AttrValue() )
```

Purpose

Use this function to associate a different index class name and index class attributes (name/values) to an existing document or folder (specified by an Item Id).

Parameters

ItemId — input

The Item Id of the document or folder which is to be changed.

ClassName

— input

VbVhlChangeItemIndex

The name of the new index class name for the item.

AttrName()

— input

An array of attribute names which correspond to the array of attribute values in *AttrValue()*. These attribute names must be defined for the specified *ClassName*.

Note: Array index 0 must contain the number of array elements.

AttrValue()

— input

An array of attribute values which correspond to the array of attribute names in *AttrName()*. These attribute values must be valid for the data type defined in index class *ClassName* for this attribute.

Note: Array index 0 must contain the number of array elements.

Guidelines for Use

The ItemId and index class name specified must exist prior to using this function. Also the attributes in the input array list must be defined for this index class and *all required* attributes of the index class *must be specified* in the array list.

Note that when specifying attribute name and value arrays, each *attribute name* array element must have a corresponding *attribute value* array element at the same array index.

Visual Basic Source Code

Function VbVhlChangeItemIndex (ItemID, ClassName, AttrName(), AttrValue())

```
' Declarations
Dim ItemObj As Object

' Setup Error handler
On Error GoTo VhlChgIndexError
u1RC = 0

' Get the search results folder
Set ItemObj = VhlApp1Obj.ItemID(ItemID)

' Update Item index class
ItemObj.Class = ClassName

' Update the Item attributes
For i = 1 To AttrName(0)
    ItemObj.KeyFields(AttrName(i)) = AttrValue(i)
Next
' Update the Items Index Class and attribute information
u1RC = ItemObj.UpdateIndex

VhlChgIndexEnd:

' Free the objects
Set ItemObj = Nothing

' Set return value to error code
VbVhlChangeItemIndex = u1RC

Exit Function

VhlChgIndexError:
```

```

' Set return code to error code
u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1ChgIndexEnd

End Function

```

VbVhICloseDocViews (Close the Document Image View Window)

Format VbVhICloseDocViews(<i>fUpdate</i>)
--

Purpose

This function closes the document which is currently displayed in the Image viewer.

Parameters

fUpdate

— input

Flag (*True* or *False*) to specify whether changes made to the document being displayed are to be saved.

Guidelines for Use

The document display window currently displayed in the Image viewer is closed after executing this function. The *fUpdate* parameter determines whether any changes (annotation, highlighting, and so forth) made to the document are saved.

Visual Basic Source Code

```

Function VbVhICloseDocViews (fUpdate)

' Declarations
Dim ImageObj As Object

' Setup Error handler
On Error GoTo Vh1CloseDocError
u1RC = 0

' Close Document being displayed
Set ImageObj = Vh1App1Obj.Image
If Not (ImageObj Is Nothing) Then
    ImageObj.CloseIt (fUpdate)
End If

Vh1CloseDocEnd:
' Set return value to error code
VbVhICloseDocViews = u1RC

Exit Function

Vh1CloseDocError:
' Set return code to error code
u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1CloseDocEnd

End Function

```

VbVhlCopyDoc (Create a Copy Of a Document)

Format

```
VbVhlCopyDoc( NewDocID, DocID, ClassName, AttrName(), AttrValue() )
```

Purpose

Use this function to create a new document and copy all the objects from an existing document into it. The new document can be set to a new index class or the default index class.

Parameters

NewDocID

— output

The Item Id for the created document is returned into this Visual Basic variable.

DocID — input

The Item Id of the original document.

ClassName

— input

The name of the new index class for the new document. If set to *NULL*, the index class will be set to *NOINDEX*.

AttrName()

— input

An array of attribute names which correspond to the array of attribute values in *AttrValue()*. These attribute names must be defined for the specified *ClassName*. Not used if *ClassName* is *NULL*.

Note: Array index 0 must contain the number of array elements.

AttrValue()

— input

An array of attribute values which correspond to the array of attribute names in *AttrName()*. These attribute values must be valid for the data type defined in index class *ClassName* for this attribute. Not used if *ClassName* is *NULL*.

Note: Array index 0 must contain the number of array elements.

Guidelines for Use

The document Item Id must be valid. If *ClassName* is not *NULL*, it must exist prior to using this function. Also the attributes in the input array list must be defined for this index class and *all required attributes that are used for uniquely indexing the index class must be specified* in the new attribute array list.

If *ClassName* is *NULL*, the index class of the new document will be set to *NOINDEX*, with attribute *Source* set to "COPY" and attributes *Name* and *Timestamp* set to the User Id and current time.

The newly created document Item ID is stored in the specified Visual Basic variable, *NewDocID*.

Visual Basic Source Code

```

Function VbVh1CopyDoc (NewDocId, DocId, ClassName, AttrName(), AttrValue())

    ' Declarations
    Dim ItemObj As Object
    DimNewItemObj As Object

    ' Setup Error handler
    On Error GoTo Vh1CopyDocError
    u1RC = 0

    ' Get the Document object
    Set ItemObj = Vh1App1Obj.ItemID(DocId)
    ' Make sure the object is a document
    If ItemObj.Type <> 1 Then
        ' Return with error - SBVI_BAD_DOCUMENT
        u1RC = 909
        GoTo Vh1CopyDocEnd
    End If

    ' Create a new document
    SetNewItemObj = Vh1App1Obj.CreateDocument("COPY")
    NewDocId =NewItemObj.ItemID

    ' Update the new document with Index Class information if provided
    If (u1RC = 0) And (ClassName <> "") Then
        ' Change the Items Index Class
        u1RC = VbVh1ChangeItemIndex(NewDocId, ClassName, AttrName(), AttrValue())
    End If

    ' Copy document base parts into new document
    i = 0
    While (u1RC = 0) And (i < ItemObj.PartCount)
        ContentClass = ItemObj.GetPartContentClass(i)
        TempFile = ItemObj.GetPartFile(i)
        u1RC =NewItemObj.AddPart(TempFile, ContentClass)
        i = i + 1
    Wend
    ' Close the original document
    RC = ItemObj.CloseParts

Vh1CopyDocEnd:

    ' Free the objects
    Set ItemObj = Nothing
    SetNewItemObj = Nothing

    ' Set return value to error code
    VbVh1CopyDoc = u1RC

    Exit Function

Vh1CopyDocError:

    ' Set return code to error code
    u1RC = Vh1ErrorObj.ReturnCode

    Resume Vh1CopyDocEnd

End Function

```

VbVhICreateFolder (Create a New Folder)

Format

```
VbVhICreateFolder( FolderId, ClassName, AttrName(), AttrValue() )
```

Purpose

Use this function to create a folder using the specified index class name and index attributes (name/values).

Parameters

FolderId

— output

The name of the Visual Basic Variable into which the created folder Item Id is stored.

ClassName

— input

The name of the index class for the folder. If NULL, the name "NOINDEX" is used.

AttrName()

— input

An array of attribute names which correspond to the array of attribute values in *AttrValue()*. These attribute names must be defined for the specified *ClassName*. Not used if *ClassName* is NULL.

Note: Array index 0 must contain the number of array elements.

AttrValue()

— input

An array of attribute values which correspond to the array of attribute names in *AttrName()*. These attribute values must be valid for the data type defined in index class *ClassName* for this attribute. Not used if *ClassName* is NULL.

Note: Array index 0 must contain the number of array elements.

Guidelines for Use

The index class name specified must be defined prior to using this function. Also the attribute names in the input array list must be defined for this index class and *all required* attributes of the index class *must be specified* in the array list.

If *ClassName* is NULL, the index class of the new folder will be set to NOINDEX, with attribute *Source* set to "CREATE" and attributes Name and Timestamp set to the User Id and current time.

The created folder Item Id is stored in the specified Visual Basic Variable, *FolderId*.

Visual Basic Source Code

```
Function VbVhICreateFolder (FolderId, ClassName, AttrName(), AttrValue())
```

```
    ' Declarations
    Dim FolderObj As Object
```



```

' Setup Error handler
On Error GoTo VhIcreFoldError
u1RC = 0

' Create the folder
Set FolderObj = VhIAppIObj.CreateFolder("CREATE")
FolderId = FolderObj.ItemID
If (u1RC = 0) And (ClassName <> "") Then
    ' Change the Items Index Class
    u1RC = VbVhIChangeItemIndex(FolderId, ClassName, AttrName(), AttrValue())
End If

VhIcreFoldEnd:

' Free the object
Set FolderObj = Nothing

' Set return value to error code
VbVhICreateFolder = u1RC

Exit Function

VhIcreFoldError:

' Set return code to error code
u1RC = VhIErrorObj.ReturnCode

Resume VhIcreFoldEnd

End Function

```

VbVhICreateFolderAddItem (Create a Folder and Add an Item)

Format

```

VbVhICreateFolderAddItem( FolderId, ItemId, ClassName, AttrName(),
AttrValue() )

```

Purpose

Use this function to create a folder using the specified index class name and index attributes (name/values). This function can also be used to add a document or folder (specified by an Item Id) to the newly created folder.

Parameters

FolderId

— output

The name of the Visual Basic Variable into which the created folder Item Id is stored.

ItemId — input

The Item Id of the document or folder which is to be added to the newly created folder.

ClassName

— input

The name of the index class for the folder. If NULL, the name "NOINDEX" is used.

AttrName()

— input

VbVh1CreateFolderAddItem

An array of attribute names which correspond to the array of attribute values in *AttrValue()*. These attribute names must be defined for the specified *ClassName*. Not used if *ClassName* is *NULL*.

Note: Array index 0 must contain the number of array elements.

AttrValue()

— input

An array of attribute values which correspond to the array of attribute names in *AttrName()*. These attribute values must be valid for the data type defined in index class *ClassName* for this attribute. Not used if *ClassName* is *NULL*.

Note: Array index 0 must contain the number of array elements.

Guidelines for Use

The Item Id and index class name specified must be defined prior to using this function. Also the attribute names in the input array list must be defined for this index class and *all required* attributes of the index class *must be specified* in the list.

If *ClassName* is *NULL*, the index class of the new folder will be set to *NOINDEX*, with attribute *Source* set to "CREATE" and attributes Name and Timestamp set to the User Id and current time.

The created folder item ID is stored in the specified Visual Basic variable, *FolderID*.

Visual Basic Source Code

```
Function VbVh1CreateFolderAddItem (FolderId, ItemID, ClassName,
                                   AttrName(), AttrValue())

    ' Declarations
    Dim FolderObj As Object
    Dim ItemObj As Object

    ' Setup Error handler
    On Error GoTo Vh1CreFoldAddError
    u1RC = 0

    ' Create the folder
    Set FolderObj = Vh1App1Obj.CreateFolder("CREATE")
    FolderId = FolderObj.ItemID

    ' Get the ItemID Object
    Set ItemObj = Vh1App1Obj.ItemID(ItemID)

    ' Put ItemId into Folder
    u1RC = ItemObj.AddToFolder(FolderObj)

    If (u1RC = 0) And (ClassName <> "") Then
        ' Change the Items Index Class
        u1RC = VbVh1ChangeItemIndex(FolderId, ClassName, AttrName(), AttrValue())
    End If

Vh1CreFoldAddEnd:
    ' Free the objects
    Set FolderObj = Nothing
    Set ItemObj = Nothing

    ' Set return value to error code
    VbVh1CreateFolderAddItem = u1RC

Exit Function
```

```

VhIcreFoldAddError:
    ' Set return code to error code
    u1RC = VhIErrorObj.ReturnCode

    Resume VhIcreFoldAddEnd

End Function

```

VbVhIDeleteItem (Delete an Item)

Format VbVhIDeleteItem(<i>ItemID</i>)
--

Purpose

Use this function to delete a document or folder as specified by the Item Id, Content Manager for iSeries.

Parameters

ItemID — input

The Item Id of the document or folder to be deleted from Content Manager for iSeries.

Guidelines for Use

The document or folder specified is physically deleted.

Visual Basic Source Code

```

Function VbVhIDeleteItem (ItemID)
    ' Declarations
    Dim ItemObj As Object

    ' Setup Error handler
    On Error GoTo VhIDeleteError
    u1RC = 0

    ' Get the ItemID Object
    Set ItemObj = VhIAppIObj.ItemID(ItemID)

    ' Delete the Item
    u1RC = ItemObj.DeleteIt

VhIDeleteEnd:
    ' Free the objects
    Set ItemObj = Nothing

    ' Set return value to error code
    VbVhIDeleteItem = u1RC

    Exit Function

VhIDeleteError:
    ' Set return value to error code
    u1RC = VhIErrorObj.ReturnCode

    Resume VhIDeleteEnd

End Function

```

VbVhIDisplayDocView (Display a Document Image)

Format

```
VbVhIDisplayDocView( DocId, fUpdate )
```

Purpose

This function displays a document image (specified by Item Id) in the Image viewer.

Parameters

DocId — input

The Item Id of the document image to be displayed.

fUpdate

— input

Flag (*True* or *False*) to specify whether changes made to the document currently displayed are to be saved.

Guidelines for Use

The document currently displayed in the Image viewer is closed before the specified new document is displayed. The *fUpdate* parameter determines whether any changes (annotation, highlighting, and so forth) made to the previous document are saved.

Visual Basic Source Code

Function VbVhIDisplayDocView (ItemID, fUpdate)

```

    ' Declarations
    Dim ItemObj As Object
    Dim ImageObj As Object

    ' Setup Error handler
    On Error GoTo VhIDispDocError
    u1RC = 0

    ' Get the Item object
    Set ItemObj = Vh1App1Obj.ItemID(ItemID)

    ' Close Document being displayed
    Set ImageObj = Vh1App1Obj.Image
    If Not (ImageObj Is Nothing) Then
        ImageObj.CloseIt (fUpdate)
    End If

    ' Display Document
    u1RC = ImageObj.OpenDocument(ItemObj)
VhIDispDocEnd:

    ' Free the object
    Set ItemObj = Nothing

    ' Set return value to error code
    VbVhIDisplayDocView = u1RC

    Exit Function

VhIDispDocError:

    ' Set return code to error code
```

```

u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1DispDocEnd

End Function

```

VbVhIDisplayVItem (Display Item Using the Client for Windows)

Format

```
VbVhIDisplayVItem( ItemID, fUpdate )
```

Purpose

The Client for Windows application is used to display the contents of a document, folder, or workbasket. A document will be displayed in the Image viewer, while folders and workbaskets are displayed in the Client for Windows main window as a separate window.

Parameters

ItemID — input

The Item ID of the Document or Folder to be displayed.

fUpdate

— input

Flag (*True* or *False*) to specify whether changes made to the document currently displayed are to be saved. This is only used if the Item Id specified is a document.

Guidelines for Use

The Document, Folder, or Workbasket information is displayed using the Client for Windows application. A document will be displayed in the Image viewer, while folders and workbaskets are displayed in the Client for Windows main window as a separate window. The *fUpdate* parameter is only used if a document is specified, this flag determines whether any changes to the currently displayed document are saved.

Visual Basic Source Code

```

Function VbVhIDisplayVItem (ItemID, fUpdate)

    ' Declarations
    Dim ItemObj As Object
    Dim ImageObj As Object
    Dim FolderObj As Object

    ' Setup Error handler
    On Error GoTo Vh1DispItemError
    u1RC = 0

    ' Get the Item object
    Set ItemObj = Vh1App1Obj.ItemID(ItemID)

    ' Find out if the item is a folder or a document
    If (ItemObj.Type = 1) Then
        ' Close Document being displayed
        Set ImageObj = Vh1App1Obj.Image
        If Not (ImageObj Is Nothing) Then
            ImageObj.CloseIt (fUpdate)
        End If
    End If
End Function

```

VbVhIDisplayVIItem

```
        End If
        ' Display Document
        ulRC = ImageObj.OpenDocument(ItemObj)
    Else
        ' Must be a folder. Display it.
        Set FolderObj = VhI DocsObj.OpenTOC(ItemObj)
    End If

VhIDispItemEnd:
    ' Free the object
    Set ItemObj = Nothing
    Set FolderObj = Nothing
    Set ImageObj = Nothing

    ' Set return value to error code
    VbVhI DisplayVIItem = ulRC

    Exit Function

VhIDispItemError:
    ' Set return code to error code
    ulRC = VhI ErrorObj.ReturnCode

    Resume VhIDispItemEnd

End Function
```

VbVhIDropFuncs (End Access to VHLPI Functions)

Format VbVhIDropFuncs()

Purpose

Use this API to end access to the Client for Windows's OLE automation interface. Any subsequent use of the VHLPI functions will fail.

Guidelines for Use

After executing this function, the Visual Basic program cannot call any VHLPI functions. To establish access to these functions, use the VbVhILoadFuncs API.

Visual Basic Source Code

```
Function VbVhIDropFuncs ()

    ' Setup Error handler
    On Error GoTo VhIDropError

    ' End access with OLE interface
    ulRC = 0
    Set VhI DocsObj = Nothing
    Set VhI ErrorObj = Nothing
    Set VhI ApplObj = Nothing

VhIDropEnd:

    ' Set return value to error code
    VbVhIDropFuncs = ulRC

    Exit Function

VhIDropError:

    ' Set return code to error code
```

```
u1RC = Err
Resume Vh1DropEnd
```

```
End Function
```

VbVhIExportDocObj (Export a Document Base Object)

Format

```
VbVhIExportDocObj( DocId, FileName, PartNum )
```

Purpose

This function creates a disk file containing a base object of a document (specified by *DocId*).

Parameters

DocId — input

The Item Id for the document whose base part is to be exported.

FileName

— input

The name (path included) of the file to create.

PartNum

— input

The part number of the base object to export. "0" represents the first base part.

Guidelines for Use

The document Item Id must be valid and the document base object must be able to be represented in a file.

Visual Basic Source Code

```
Function VbVhIExportDocObj (DocId, Filename, PartNum)
```

```
    ' Declarations
    Dim DocObj As Object

    ' Setup Error handler
    On Error GoTo VhIExportDocError
    u1RC = 0

    ' Get the document object
    Set DocObj = Vh1App1Obj.ItemID(DocId)

    ' Copy document base part into file
    TempFile = DocObj.GetPartFile(PartNum)
    Name TempFile As Filename
    ' Close the document
    RC = DocObj.CloseParts
```

```
VhIExportDocEnd:
```

```
    ' Free the object
    Set DocObj = Nothing

    ' Set return value to error code
    VbVhIExportDocObj = u1RC
```

VbVhlExportDocObj

```
Exit Function

VhlExportDocError:

    ' Set return code to error code
    u1RC = VhlErrorObj.ReturnCode

Resume VhlExportDocEnd

End Function
```

VbVhlGetVIUserID (Get the Logon User ID)

Format
VbVhlGetVIUserID()

Purpose

Use this function to return the logged—on User Id.

Guidelines for Use

A NULL User ID is returned in case of an error, say for example, no logon session exists.

Visual Basic Source Code

```
Function VbVhlGetVIUserID ()

    ' Setup Error handler
    On Error GoTo VhlGetUserError
    u1RC = 0

    ' Set return value to UserId
    VbVhlGetVIUserID = VhlApp1Obj.User

VhlGetUserEnd:

Exit Function

VhlGetUserError:

    ' Set return code to error code
    VbVhlGetVIUserID = VhlErrorObj.ReturnCode

Resume VhlGetUserEnd

End Function
```

VbVhlImportDocObj (Import a Document Base Object)

Format
VbVhlImportDocObj(DocId, FileName, ContentClass, ClassName, AttrName(), AttrValue())

Purpose

This function creates a document base object from a disk file format of the document base part.

Parameters

DocId — output

The name of the Visual Basic Variable into which the document Item Id is stored.

FileName

— input

The name (path included) of the file containing the document base part (file extension is included).

ContentClass

— input

The content class name for the file.

ClassName

— input

The name of the index class for the document. If NULL or not specified, the name "NOINDEX" is used.

AttrName()

— input

An array of attribute names which correspond to the array of attribute values in *AttrValue()*. These attribute names must be defined for the specified *ClassName*. Not used if *ClassName* is NULL.

Note: Array index 0 must contain the number of array elements.

AttrValue()

— input

An array of attribute values which correspond to the array of attribute names in *AttrName()*. These attribute values must be valid for the data type defined in index class *ClassName* for this attribute. Not used if *ClassName* is NULL.

Note: Array index 0 must contain the number of array elements.

Guidelines for Use

The index class name specified must exist prior to using this function. Also the attribute names in the input array list must be defined for this index class and *all required* attributes of the index class *must be specified* in the list.

The created document Item Id is stored in the specified Visual Basic Variable, *DocId*.

Visual Basic Source Code

```
Function VbVhlImportDocObj (DocId, Filename, ContentClass,
                          ClassName, AttrName(), AttrValue())

    ' Declarations
    Dim DocObj As Object

    ' Setup Error handler
    On Error GoTo VhlImportDocError
    ulRC = 0

    ' Create the document and add the file
    Set DocObj = VhlApp1Obj.CreateDocument("IMPORT")
```

VbVhlImportDocObj

```
DocId = DocObj.ItemID
u1RC = DocObj.AddPart(Filename, ContentClass)
If (u1RC = 0) And (ClassName <> "") Then
    ' Change the Items Index Class
    u1RC = VbVhlChangeItemIndex(DocId, ClassName, AttrName(), AttrValue())
End If

VhlImportDocEnd:

    ' Free the object
    Set DocObj = Nothing

    ' Set return value to error code
    VbVhlImportDocObj = u1RC

Exit Function

VhlImportDocError:

    ' Set return code to error code
    u1RC = VhlErrorObj.ReturnCode

Resume VhlImportDocEnd

End Function
```

VbVhlListContClasses (List all Content Classes)

Format VbVhlListContClasses(CCList())
--

Purpose

Use this function to list all content classes.

Parameters

CCList()

— output

The name of the Visual Basic variable into which is stored the list of all content classes. This Visual Basic variable name will be an array variable with the index count (number of content classes returned) stored in *CCList(0)*. The format of the Visual Basic array is as follows:

CCList(0)— # of content classes

CCList(n)— Content Class name *n*

Guidelines for Use

This function lists both the IBM-defined Content Classes and the user-defined content classes.

Visual Basic Source Code

```
Function VbVhlListContClasses (CCList())

    ' Declarations
    Dim i, u1Start, u1End, u1Len, u1TotLen As Long

    ' Setup Error handler
    On Error GoTo VhlContListError
    u1RC = 0

    ' Get the list of Cont Classes
```

```

strRet = Vh1App1Obj.ContentClassList(";")
u1TotLen = Len(strRet)

' Add Cont classes to List array
i = 0
ReDim CCList(1)
CCList(0) = 0
u1Start = 1
Do
    ' Each name separated by a ";"
    u1End = InStr(u1Start, strRet, ";")
    If (u1End = 0) Then
        u1End = u1TotLen + 1
    End If
    u1Len = u1End - u1Start

    ' Set next array variable to Cont Class name
    i = i + 1
    ReDim Preserve CCList(i + 1)
    CCList(i) = Mid$(strRet, u1Start, u1Len)

    ' Setup for next loop
    u1Start = u1End + 1
Loop Until (u1Start >= u1TotLen)

' Set total number of Cont Classes in array
CCList(0) = i

Vh1ContListEnd:

' Set return value to error code
VbVhListContClasses = u1RC

Exit Function

Vh1ContListError:

' Set return code to error code
u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1ContListEnd

End Function

```

VbVhListFolderItems (List Folder Contents)

Format

```
VbVhListFolderItems( ItemList(), FolderID, IndexClass() )
```

Purpose

Use this function to list all document and folder Item IDs contained in a folder (specified by the folder's Item ID), and matching the optional index classes array specification.

Parameters

ItemList()

— output

The name of the Visual Basic Variable into which is stored the list of documents and folders contained in the specified folder's table of contents and also matching the optional index classes. This Visual Basic Variable

VbVhlListFolderItems

name will be an array variable with the index count (number of Item IDs returned) stored in *ItemList(0,0)*, and for each returned item, a structure of three Visual Basic array elements are created, such as:

ItemList(n,1)— Item ID

ItemList(n,2)— Item Type

——(1)*Document*

——(2)*Folder*

——(?)*Unknown*

ItemList(n,3)— Index class

FolderID

— input

The Item Id of the folder to list.

IndexClass()

— input

Optional index classes to filter the items returned. If no elements specified, all the items in the Folder's table of contents will be returned, regardless of its index class.

Note: Array index 0 must contain the number of array elements in the list.

Guidelines for Use

The folder Item Id must exist prior to this call. This function can also be used to list the contents of a workbasket.

Visual Basic Source Code

Function VbVhlListFolderItems (ItemList(), FolderId, IndexClass())

```
' Declarations
Dim FolderObj As Object
Dim ContentObj As Object
Dim ulTOCCnt, ulStart, ulEnd, ulLen, ulTotLen As Long

' Setup Error handler
On Error GoTo VhlLstFldError
ulRC = 0

' Get the Folder Object
Set FolderObj = VhlApp1Obj.ItemID(FolderId)

' Setup return array based on size of folder
ulTOCCnt = FolderObj.TOCCount
ReDim ItemList(ulTOCCnt + 1, 4)
ItemList(0, 0) = 0

' Get the list of Item Objects in the Folder
j = 1
For i = 1 To ulTOCCnt
    Set ContentObj = FolderObj.GetTOCItem(i - 1)
    ItemList(j, 0) = 3
    ItemList(j, 1) = ContentObj.ItemID
    ItemList(j, 2) = ContentObj.Type
    ItemList(j, 3) = ContentObj.Class
    Set ContentObj = Nothing
    ' Check if Index Class filter was provided
    Found = False
    If IndexClass(0) <> 0 Then
        For k = 1 To IndexClass(0)
            If IndexClass(k) = ItemList(j, 3) Then
```

```

        Found = True
        Exit For
    End If
Next k
Else
    Found = True
End If
' Only send back Items found in Index Class list
If Found Then
    ItemList(0, 0) = j
    j = j + 1
End If
Next i

Vh1LstFldEnd:

' Free the objects
Set ContentObj = Nothing
Set FolderObj = Nothing

' Set return value to error code
VbVhListFolderItems = ulRC

Exit Function

Vh1LstFldError:

' Set return code to error code
ulRC = Vh1ErrorObj.ReturnCode

Resume Vh1LstFldEnd

End Function

```

VbVhListFolderItemsAttr (List Folder Contents and Their Attributes)

Format

VbVhListFolderItemsAttr(*ItemList()*, *FolderId*)

Purpose

Use this function to list all document and folder Item Ids contained in a folder (specified by the folder's Item Id).

Parameters

ItemList()

— output

The name of the Visual Basic Variable into which is stored the list of documents and folders contained in the specified folder's table of contents. This Visual Basic Variable name will be an array variable with the index count (number of Item IDs returned) stored in *ItemList(0,0)*, and for each returned item, a structure of Visual Basic array elements are created, such as:

ItemList(n,0)— size of array

ItemList(n,1)— Item ID

ItemList(n,2)— Item Type

—(1)Document

VbVhlListFolderItemsAttr

——(2)Folder

——(?)Unknown

ItemList(n,3)— Index class

ItemList(n,3+m) — Attribute name *m*

ItemList(n,3+m) — Attribute value *m*

FolderId

— input

The Item Id of the folder to list.

Guidelines for Use

The folder Item Id must exist prior to this call. This function can also be used to list the contents of a workbasket.

Visual Basic Source Code

Function VbVhlListFolderItemsAttr (ItemList(), FolderId)

```
' Declarations
Dim FolderObj As Object
Dim ContentObj As Object
Dim ulTOCCnt, ulStart, ulEnd, ulLen, ulTotLen As Long

' Setup Error handler
On Error GoTo VhlLstFldAttrError
ulRC = 0

' Get the Folder Object
Set FolderObj = VhlApp1Obj.ItemID(FolderId)

' Setup return array based on size of folder
ulTOCCnt = FolderObj.TOCCount
ReDim ItemList(ulTOCCnt + 1, 4)
ItemList(0, 0) = 0

' Get the list of Item Objects in the Folder
For i = 1 To ulTOCCnt
    Set ContentObj = FolderObj.GetTOCItem(i - 1)
    ItemList(i, 1) = ContentObj.ItemID
    ItemList(i, 2) = ContentObj.Type
    ItemList(i, 3) = ContentObj.Class
    ItemList(0, 0) = i
    ItemList(i, 0) = 3

    ' Get the list of Index Class attributes
    strRet = VhlApp1Obj.ClassKeyFieldList(ContentObj.Class, ";")
    ulTotLen = Len(strRet)
    j = 3
    ulStart = 1
    ' Add attributes to List array
    Do
        ' Each name separated by a ";"
        ulEnd = InStr(ulStart, strRet, ";")
        If (ulEnd = 0) Then
            ulEnd = ulTotLen + 1
        End If
        ulLen = ulEnd - ulStart
        AttrName = Mid$(strRet, ulStart, ulLen)

        ' Set next array variables to attribute name and value
        j = j + 1
        ReDim Preserve ItemList(i, j + 2)
        ItemList(i) = AttrName
        j = j + 1
    Loop
```

```

        ItemList(i, j) = ContentObj.KeyFields(AttrName)

        ' Setup for next loop
        ulStart = ulEnd + 1
    Loop Until (ulStart >= ulTotLen)

    ' Reset total number of variables in array
    ItemList(i, 0) = j
    ' Free the current Item object
    Set ContentObj = Nothing

Next i

Vh1LstFldAttrEnd:

    ' Free the objects
    Set ContentObj = Nothing
    Set FolderObj = Nothing

    ' Set return value to error code
    VbVhlListFolderItemsAttr = ulRC

Exit Function

Vh1LstFldAttrError:

    ' Set return code to error code
    ulRC = Vh1ErrorObj.ReturnCode

Resume Vh1LstFldAttrEnd

End Function

```

VbVhlListIndexClassAttr (List All Attributes Of an Index Class)

Format

```
VbVhlListIndexClassAttr( AttrList(), ClassName )
```

Purpose

This function lists all the attributes of a specified Index Class.

Parameters

AttrList()

— output

The name of the Visual Basic variable into which is stored the list of all the attribute names of the specified Index Class name. This Visual Basic variable name will be a array variable with the index count (number of attributes returned) stored in *AttrList(0)*. The format of the Visual Basic array is as follows:

AttrList(0)— # of attributes

AttrList(n)— Attribute Name *n*

ClassName

— input

The Index class name for which all attribute names are to be listed.

VbVhListIndexClassAttr

Guidelines for Use

This function lists only attributes of an Index Class name for which the user has access.

Visual Basic Source Code

```
Function VbVhListIndexClassAttr (AttrList(), ClassName)
```

```
    ' Declarations
    Dim i, ulStart, ulEnd, ulLen, ulTotLen As Long

    ' Setup Error handler
    On Error GoTo Vh1ClassAttrError
    ulRC = 0

    ' Get the list of Index Class attributes
    strRet = Vh1App1Obj.ClassKeyFieldList(ClassName, ";")
    ulTotLen = Len(strRet)

    ' Add attributes to List array
    i = 0
    ReDim AttrList(1)
    AttrList(0) = 0
    ulStart = 1
    Do
        ' Each name separated by a ";"
        ulEnd = InStr(ulStart, strRet, ";")
        If (ulEnd = 0) Then
            ulEnd = ulTotLen + 1
        End If
        ulLen = ulEnd - ulStart

        ' Set next array variable to attribute name
        i = i + 1
        ReDim Preserve AttrList(i + 1)
        AttrList(i) = Mid$(strRet, ulStart, ulLen)

        ' Setup for next loop
        ulStart = ulEnd + 1
    Loop Until (ulStart >= ulTotLen)

    ' Set total number of attributes in array
    AttrList(0) = i

Vh1ClassAttrEnd:

    ' Set return value to error code
    VbVhListIndexClassAttr = ulRC

    Exit Function

Vh1ClassAttrError:

    ' Set return code to error code
    ulRC = Vh1ErrorObj.ReturnCode

    Resume Vh1ClassAttrEnd

End Function
```


VbVhlListIndexClasses (List all Index Classes)

Format

```
VbVhlListIndexClasses( IxClassList() )
```

Purpose

Use this function to list all user accessible Index Classes.

Parameters

IxClassList()

— output

The name of the Visual Basic variable into which is stored the returned Index Classes. This Visual Basic variable name will be an array variable with the index count (number of index classes returned) stored in *IxClassList(0)*. The format of the Visual Basic array is as follows:

IxClassList(0)— # of index classes

IxClassList(n)— Index Class name *n*

Visual Basic Source Code

```
Function VbVhlListIndexClasses (IxClassList())
```

```
    ' Declarations
    Dim i, ulStart, ulEnd, ulLen, ulTotLen As Long

    ' Setup Error handler
    On Error GoTo VhlClassListError
    ulRC = 0

    ' Get the list of Index Classes
    strRet = VhlApp1Obj.ClassList(";")
    ulTotLen = Len(strRet)

    ' Add Index classes to List array
    i = 0
    ReDim IxClassList(1)
    IxClassList(0) = 0
    ulStart = 1
    Do
        ' Each name separated by a ";"
        ulEnd = InStr(ulStart, strRet, ";")
        If (ulEnd = 0) Then
            ulEnd = ulTotLen + 1
        End If
        ulLen = ulEnd - ulStart

        ' Set next array variable to Index Class name
        i = i + 1
        ReDim Preserve IxClassList(i + 1)
        IxClassList(i) = Mid$(strRet, ulStart, ulLen)

        ' Setup for next loop
        ulStart = ulEnd + 1
    Loop Until (ulStart >= ulTotLen)

    ' Set total number of Index Classes in array
    IxClassList(0) = i
```

```
VhlClassListEnd:
```

VbVhListIndexClasses

```
' Set return value to error code
VbVhListIndexClasses = u1RC

Exit Function

Vh1ClassListError:

' Set return code to error code
u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1ClassListEnd

End Function
```

VbVhListItemCC (List a Base Object's Content Class)

Format

```
VbVhListItemCC( ItemCC, ItemId, PartNum )
```

Purpose

This function lists the Content Class associated with a base object of the specified Item Id.

Parameters

ItemCC

— output

The name of the Visual Basic Variable into which is stored the returned Content Class name.

ItemId — input

The Item Id.

PartNum

— input

The part number of the document to return content class information. "0" represents the first base part.

Guidelines for Use

The Item Id must exist prior to this call.

Visual Basic Source Code

```
Function VbVhListItemCC (ItemCC, ItemId, PartNum)

' Declarations
Dim ItemObj As Object

' Setup Error handler
On Error GoTo Vh1ItemCCError
u1RC = 0

' Get the Item object
Set ItemObj = Vh1App1Obj.ItemID(ItemId)

' Copy content class of document base part
ItemCC = ItemObj.GetPartContentClass(PartNum)

Vh1ItemCCEnd:
```

```

' Free the object
Set ItemObj = Nothing

' Set return value to error code
VbVhListItemCC = u1RC

Exit Function

VhItemCCErrror:

' Set return code to error code
u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1ItemCCEnd

End Function

```

VbVhListItemInfo (List an Item's Index Class and Attribute Information)

Format

```
VbVhListItemInfo( ItemInfo(), ItemId )
```

Purpose

This function lists information about the specified Item Id such as — item type, index class type, and attribute names and values.

Parameters

ItemInfo

— output

The name of the Visual Basic Variable into which is stored the item information. This Visual Basic Variable name will be an array variable with the index count (size of the array variable) stored in *ItemInfo(0)*, and a structure such as:

ItemInfo(0)— array size

ItemInfo(1)— Item ID

ItemInfo(2)— Item Type

—(1)Document

—(2)Folder

—(3)Workbasket

—(?)Unknown

ItemInfo(3)— Index class

ItemInfo(3+m) — Attribute name *m*

ItemInfo(3+m) — Attribute value *m*

ItemId — input

The Item Id.

Guidelines for Use

The Item Id must exist prior to this call. Index class and attribute information do not pertain to workbasket items.

Visual Basic Source Code

```
Function VbVhListItemInfo (ItemList(), ItemID)

    ' Declarations
    Dim ItemObj As Object
    Dim ulTOCCnt, ulStart, ulEnd, ulLen, ulTotLen As Long

    ' Setup Error handler
    On Error GoTo VhListItemInfoError
    ulRC = 0

    ' Get the Item Object
    Set ItemObj = Vh1AppObj.ItemID(ItemID)

    ' Get the list of Item Objects in the Folder
    ReDim ItemList(10)
    ItemList(0) = 3
    ItemList(1) = ItemObj.ItemID
    ItemList(2) = ItemObj.Type
    ItemList(3) = ItemObj.Class

    ' Workbaskets don't have attributes
    If ItemList(2) > 2 Then
        GoTo VhListItemInfoEnd
    End If

    ' Get the list of Index Class attributes
    strRet = Vh1AppObj.ClassKeyFieldList(ItemObj.Class, ";")
    ulTotLen = Len(strRet)
    i = 3
    ulStart = 1
    ' Add attributes to List array
    Do
        ' Each name separated by a ";"
        ulEnd = InStr(ulStart, strRet, ";")
        If (ulEnd = 0) Then
            ulEnd = ulTotLen + 1
        End If
        ulLen = ulEnd - ulStart
        AttrName = Mid$(strRet, ulStart, ulLen)

        ' Set next array variables to attribute name and value
        i = i + 1
        ReDim Preserve ItemList(i + 2)
        ItemList(i) = AttrName
        i = i + 1
        ItemList(i) = ItemObj.KeyFields(AttrName)

        ' Setup for next loop
        ulStart = ulEnd + 1
    Loop Until (ulStart >= ulTotLen)

    ' Set total number of variables in array
    ItemList(0) = i

VhListItemInfoEnd:

    ' Free the objects
    Set ItemObj = Nothing

    ' Set return value to error code
    VbVhListItemInfo = ulRC

    Exit Function

VhListItemInfoError:
```

```

' Set return code to error code
u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1ListInfoEnd

End Function

```

VbVhListWBItems (List Workbasket Contents)

Format

```
VbVhListWBItems( ItemList(), WorkBasket )
```

Purpose

This function lists all the document and folder Item Ids that are contained in the workbasket (specified by name).

Parameters

ItemList()

— output

The name of the Visual Basic Variable into which the Item Ids are stored. This Visual Basic Variable name will be an array variable with the number of items stored in *ItemList(0)*, and the Item Ids in *ItemList(1)* through *ItemList(n)*.

WorkBasket

— input

The workbasket name.

Guidelines for Use

The workbasket name must be valid.

Visual Basic Source Code

```

Function VbVhListWBItems (ItemList(), WBItemID)

' Declarations
Dim WBObj As Object
Dim ContentObj As Object
Dim ulTOCCnt As Long

' Setup Error handler
On Error GoTo Vh1LstWBItemError
u1RC = 0

' Get the WB Object
Set WBObj = Vh1App1Obj.ItemID(WBItemID)

' Setup return array based on size of WB
ulTOCCnt = WBObj.TOCCnt
ReDim ItemList(ulTOCCnt + 1)
ItemList(0) = 0

' Get the list of Item Objects in the WB
j = 1
For i = 1 To ulTOCCnt
    Set ContentObj = WBObj.GetTOCItem(i - 1)
    ItemList(j) = ContentObj.ItemID
    Set ContentObj = Nothing
    ItemList(0) = j
    j = j + 1

```

VbVhListWBItems

```
Next i
Vh1LstWBItemEnd:
    ' Free the objects
    Set ContentObj = Nothing
    Set WBObj = Nothing

    ' Set return value to error code
    VbVh1ListWBItems = u1RC

Exit Function

Vh1LstWBItemError:
    ' Set return code to error code
    u1RC = Vh1ErrorObj.ReturnCode

Resume Vh1LstWBItemEnd

End Function
```

VbVhListWorkBaskets (List All Workbasket Names)

Format

```
VbVh1ListWorkBaskets( WkBasketList() )
```

Purpose

Use this function to list all the workbasket names and descriptions.

Parameters

WkBasketList()

— output

The name of the Visual Basic variable into which is stored the list of defined workbasket names. This Visual Basic variable name will be an array variable with the index count (number of workbaskets returned) stored in *WkBasketList(0)*, and the workbasket names stored in *WkBasketList(1)* through *WkBasketList(n)*.

Visual Basic Source Code

```
Function VbVh1ListWorkBaskets (WBList())

    ' Declarations
    Dim i, u1Start, u1End, u1Len, u1TotLen As Long

    ' Setup Error handler
    On Error GoTo Vh1ListWBError
    u1RC = 0

    ' Get the list of WorkBaskets
    strRet = Vh1App1Obj.WorkBasketList(";")
    u1TotLen = Len(strRet)

    ' Add Index classes to List array
    i = 0
    ReDim WBList(1)
    WBList(0) = 0
    u1Start = 1
    Do
        ' Each name separated by a ";"
```

```

    u1End = InStr(u1Start, strRet, ";")
    If (u1End = 0) Then
        u1End = u1TotLen + 1
    End If
    u1Len = u1End - u1Start

    ' Set next array variable to Index Class name
    i = i + 1
    ReDim Preserve WBLList(i + 1)
    WBLList(i) = Mid$(strRet, u1Start, u1Len)

    ' Setup for next loop
    u1Start = u1End + 1
    Loop Until (u1Start >= u1TotLen)

    ' Set total number of Index Classes in array
    WBLList(0) = i

VhlListWBEnd:

    ' Set return value to error code
    VbVhlListWorkBaskets = u1RC

    Exit Function

VhlListWBError:

    ' Set return code to error code
    u1RC = VhlErrorObj.ReturnCode

    Resume VhlListWBEnd

End Function

```

VbVhlLoadFuncs (Get Access to VHLPI Functions)

Format VbVhlLoadFuncs()
--

Purpose

Use this function to gain access to the VHLPI functions for Visual Basic. This allows the Visual Basic program to call these functions.

Guidelines for Use

After executing this function, the Visual Basic program can call any VHLPI function. To terminate access to these functions, use the VbVhlDropFuncs function.

Visual Basic Source Code

```

Function VbVhlLoadFuncs ()

    ' Setup Error handler
    On Error GoTo VhlLoadError
    u1RC = 0

    ' Get the application object
    Set VhlApp1Obj = CreateObject("Vic.Application")

    ' Setup Global Application Objects
    Set VhlDocsObj = VhlApp1Obj.Documents
    Set VhlErrorObj = VhlApp1Obj.Error

VhlLoadEnd:

```

VbVhlLoadFuncs

```
' Set return value to error code
VbVhlLoadFuncs = u1RC

Exit Function

VhlLoadError:

' Set return code to error code
u1RC = Err
Resume VhlLoadEnd

End Function
```

VbVhlLogoff (End Access to IBM Content Manager for iSeries)

Format VbVhlLogoff()

Purpose

Use this API to end access and close the Client for Windows. Any subsequent use of the VHLPI functions will fail.

Guidelines for Use

After executing this function, the Client for Windows will be closed and no Visual Basic program can call VHLPI functions. To establish access to these functions, use the VbVhlLogon API.

Visual Basic Source Code

```
Function VbVhlLogoff ()

' Setup Error handler
On Error GoTo VhlLogoffError

' Logoff from the system
u1RC = 0
VhlApp1Obj.Quit
Set VhlDocsObj = Nothing
Set VhlErrorObj = Nothing
Set VhlApp1Obj = Nothing

VhlLogoffEnd:

' Set return value to error code
VbVhlLogoff = u1RC

Exit Function

VhlLogoffError:

' Set return code to error code
u1RC = Err
Resume VhlLogoffEnd

End Function
```


VbVhlLogon (Get Access to IBM Content Manager for iSeries)

Format VbVhlLogon()

Purpose

Use this function to logon and gain access to the VHLPI functions for Visual Basic. This allows the Visual Basic program to call these functions.

Guidelines for Use

After executing this function, the Visual Basic program can call any VHLPI function. To logoff and close the Client for Windows, use the VbVhlLogoff function. To simply terminate access to these functions, use the VbVhlDropFuncs function.

Visual Basic Source Code

Function VbVhlLogon (UserId, Password, LibServer)

```

' Setup Error handler
On Error GoTo VhlLogonError
u1RC = 0

' Get the application object
Set VhlApp1Obj = CreateObject("Vic.Application")

' Set logon information
VhlApp1Obj.User = UserId
VhlApp1Obj.Server = LibServer
VhlApp1Obj.Password = Password

' Display the Logon screen and Log onto the system
u1RC = VhlApp1Obj.Logon
If (u1RC = 0) Then
    ' Setup Global Application Objects
    Set VhlDocsObj = VhlApp1Obj.Documents
    Set VhlErrorObj = VhlApp1Obj.Error
Else
    ' Release application object
    Set VhlApp1Obj = Nothing
End If

VhlLogonEnd:

' Set return value to error code
VbVhlLogon = u1RC

Exit Function
VhlLogonError:

' Set return code to error code
u1RC = Err
Resume VhlLogonEnd

End Function

```

VbVhlRemoveFolderItem (Remove an Item From a Folder)

Format

```
VbVhlRemoveFolderItem( ItemId, FolderId )
```

Purpose

This function removes a document or folder (specified by Item Id) from a folder (specified by the folder's Item Id).

Parameters

ItemId — input

The Item Id for the document or folder to be removed.

FolderId

— input

The Item Id for the folder.

Guidelines for Use

The document or folder specified is NOT physically deleted. It is simply disassociated with the folder.

Visual Basic Source Code

Function VbVhlRemoveFolderItem (ItemID, FolderId)

```

    ' Declarations
    Dim ItemObj As Object
    Dim FolderObj As Object

    ' Setup Error handler
    On Error GoTo VhlRemFolderError
    u1RC = 0

    ' Get the Folder Object
    Set FolderObj = VhlApp1Obj.ItemID(FolderId)

    ' Get the ItemID Object
    Set ItemObj = VhlApp1Obj.ItemID(ItemID)

    ' Put ItemId into Folder
    u1RC = ItemObj.RemoveFromFolder(FolderObj)

VhlRemFolderEnd:

    ' Free the objects
    Set ItemObj = Nothing
    Set FolderObj = Nothing

    ' Set return value to error code
    VbVhlRemoveFolderItem = u1RC

    Exit Function

VhlRemFolderError:

    ' Set return value to error code
    u1RC = VhlErrorObj.ReturnCode

    Resume VhlRemFolderEnd

End Function

```

VbVhIScanDoc (Scan Documents)

Format VbVhIScanDoc()
--

Purpose

This function invokes the Scan facility. This enables the user to scan images and create new documents. The created document's Item Ids will **not** be returned when the user closes the Scan window.

Guidelines for Use

The user interacts with the Scan facility to perform the work. Hence the user controls how and when documents are created via his commands to the Scan facility.

Visual Basic Source Code

```
Function VbVhIScanDoc ()

    ' Setup Error handler
    On Error GoTo VhIScanDocError
    u1RC = 0

    ' Scan some documents
    VhIApp1Obj.OpenScan

VhIScanDocEnd:

    Exit Function

VhIScanDocError:

    ' Set return code to error code
    VbVhIScanDoc = VhIErrorObj.ReturnCode

    Resume VhIScanDocEnd

End Function
```

VbVhISearchAdv (Advanced Search for Items)

Format VbVhISearchAdv(ItemList(), ClassName, Criteria, TypeFilter, WIPFilter, SuspendFilter)

Purpose

This function lists all the Item Ids matching the supplied search criteria. The list of returned Item Ids can be filtered based upon the values supplied for the various filter parameters.

Parameters

ItemList()
 — output

The name of the Visual Basic Variable into which the document list of Item Ids is stored. This Visual Basic Variable name will be an array variable with the number of items stored in *ItemList(0)*, and the Item Ids in *ItemList(1)* through *ItemList(n)*.

ClassName

— input

The name of the index class.

Criteria

— input

The search criteria. See "Guidelines for Use."

TypeFilter

— input

The type value of item to search for. Valid values are —

- 1(*SIM_DOCUMENT*)
- 2(*SIM_FOLDER*)
- *other(SIM_FOLDER_DOC)*

WIPFilter

— input

The Work In Progress status for items to return. The values for WIP status can be ORed together if more than one criteria is desired. Valid values are —

- 1(*OIM_ITEMS_NOT_IN_WORKFLOW*)
- 2(*OIM_CURRENT_WORKFLOW_ITEMS*)
- 4(*OIM_CANCELLED_WORKFLOW_ITEMS*)
- 8(*OIM_COMPLETED_WORKFLOW_ITEMS*)

SuspendFilter

— input

The suspension status for items to return. Valid values are —

- 1(*OIM_ITEMS_NOT_SUSPENDED*)
- 2(*OIM_ITEMS_SUSPENDED*)
- *other(OIM_ITEMS_ALL)*

Guidelines for Use

The specified index class name must exist prior to using this function. Also the Attribute Ids in the search specification must be defined for this index class.

The syntax of the search criteria is — "Attribute Operator Value" where

- *Attribute* is the Id of an attribute which must be defined in IBM Content Manager for iSeries. This attribute Id is in the format, Annn, where nnn is the attribute number.
- *Operator* is a text string representing the operation where valid "Operator" values are EQ, ==, LEQ, <=, GEQ, >=, LT, <, GT, >, NEQ, <>, IN, NOTIN, LIKE, NOTLIKE, BETWEEN, NOTBETWEEN.
- *Value* can be text, numbers, or the word *NULL*. The "Value" text can also contain the character '%' which matches any characters or the character '_' which matches any single character. Examples of valid "Operator Value" search criteria are:
 - "LIKE E%"

- "< 123"
- "==" NULL"

The system uses the search criteria to find any matching Item Ids in the database, via a dynamic SQL query.

Visual Basic Source Code

```
Function VbVh1SearchAdv (ItemList(), ClassName, Criteria,
                        TypeFilter, WIPFilter, SuspendFilter)

    ' Declarations
    Dim FolderObj As Object
    Dim ContentObj As Object
    Dim ulTOCCnt, ulStart, ulEnd, ulLen, ulTotLen As Long

    ' Setup Error handler
    On Error GoTo Vh1SearchAdvError
    ulRC = 0

    ' Get the search results folder
    Set FolderObj = Vh1App1Obj.Search(ClassName, Criteria,
                                     TypeFilter, WIPFilter, SuspendFilter)

    ' Setup return array based on size of folder
    ulTOCCnt = FolderObj.TOCCnt
    ReDim ItemList(ulTOCCnt + 1)
    ItemList(0) = 0

    ' Get the list of Item Objects in the Folder
    For i = 1 To ulTOCCnt
        Set ContentObj = FolderObj.GetTOCItem(i - 1)
        ItemList(i) = ContentObj.ItemID
        Set ContentObj = Nothing
        ItemList(0) = i
    Next

Vh1SearchAdvEnd:
    ' Free the objects
    Set ContentObj = Nothing
    Set FolderObj = Nothing

    ' Set return value to error code
    VbVh1SearchAdv = ulRC

    Exit Function

Vh1SearchAdvError:
    ' Set return code to error code
    ulRC = Vh1ErrorObj.ReturnCode

    Resume Vh1SearchAdvEnd

End Function
```

VbVh1SearchItem (Search for Items)

Format

```
VbVh1SearchItem( ItemList(), ClassName, Criteria )
```

Purpose

This function lists all the Item Ids of the specified index class name, which contain attribute names/values matching the supplied search criteria.

Parameters

ItemList()

— output

The name of the Visual Basic Variable into which the document list of Item Ids is stored. This Visual Basic Variable name will be an array variable with the number of items stored in *ItemList(0)*, and the Item Ids in *ItemList(1)* through *ItemList(n)*.

ClassName

— input

The name of the index class.

Criteria

— input

The search criteria. See "Guidelines for Use" on page 240.

Guidelines for Use

The specified index class name must exist prior to using this function. Also the Attribute Ids in the search specification must be defined for this index class.

The syntax of the search criteria is — "Attribute Operator Value" where

- *Attribute* is the Id of an attribute which must be defined. This attribute Id is in the format, Annn, where nnn is the attribute number.
- *Operator* is a text string representing the operation where valid "Operator" values are EQ, ==, LEQ, <=, GEQ, >=, LT, <, GT, >, NEQ, <>, IN, NOTIN, LIKE, NOTLIKE, BETWEEN, NOTBETWEEN.
- *Value* can be text, numbers, or the word *NULL*. The "Value" text can also contain the character '%' which matches any characters or the character '_' which matches any single character. Examples of valid "Operator Value" search criteria are:
 - "LIKE E%"
 - "< 123"
 - "==" NULL"

The system uses the search criteria to find any matching Item Ids in the database, via a dynamic SQL query.

Visual Basic Source Code

```
Function VbVh1SearchItem (ItemList(), ClassName, Criteria)
```

```
    ' Declarations  
    Dim FolderObj As Object  
    Dim ContentObj As Object  
    Dim ulTOCCnt, ulStart, ulEnd, ulLen, ulTotLen As Long
```

```
    ' Setup Error handler  
    On Error GoTo Vh1SearchError  
    ulRC = 0
```

```
    ' Get the search results folder  
    Set FolderObj = Vh1App1Obj.Search(ClassName, Criteria)
```

```

' Setup return array based on size of folder
ulTOCCnt = FolderObj.TOCCnt
ReDim ItemList(ulTOCCnt + 1)
ItemList(0) = 0

' Get the list of Item Objects in the Folder
For i = 1 To ulTOCCnt
    Set ContentObj = FolderObj.GetTOCItem(i - 1)
    ItemList(i) = ContentObj.ItemID
    Set ContentObj = Nothing
    ItemList(0) = i
Next

VhlSearchEnd:

' Free the objects
Set ContentObj = Nothing
Set FolderObj = Nothing

' Set return value to error code
VbVhlSearchItem = ulRC

Exit Function

VhlSearchError:

' Set return code to error code
ulRC = VhlErrorObj.ReturnCode

Resume VhlSearchEnd

End Function

```

Chapter 7. Content Manager for iSeries Programming Interface APIs on the Server

Server Versions of the Content Manager for iSeries Client APIs

The Content Manager for iSeries client APIs are also available as equivalent server APIs for the Content Manager for iSeries. Sample programs using some of these APIs are available in COBOL, RPG and C. For information, refer to the sample programs in the QSMPSRC source file in your QVI library. Also provided are sample data structures in the following source files: QVIRPGCPY, QVICBLCPY and H. Create your custom modules using ILE C/400®, ILE COBOL/400®, ILE RPG/400®, or VisualAge/400. Then create a program binding your new modules with service program QVI-API.

The Content Manager for iSeries Application Programming Guide & Reference (SC23–4586) may be used as a reference, noting these differences:

- Pointers are 16 bytes on the Content Manager for iSeries, so all pointers returned in the RCSTRUCT are accessed through pParam2 instead of ulParam1 and ulParam2.
- When running the APIs on the Content Manager for iSeries, the server code is run in the same job space as the application calling the APIs – a separate job is not started.
- Only image data accessible on the Content Manager for iSeries can be opened through **SimLibOpenObject**.
- Two workstation APIs do not have equivalent server versions. **Sim400SendReceive** and **Sim400ConvertCodepage** are available on the workstation only.
- The VI400TST program is available to run on either the Content Manager for iSeries of the workstation to verify the behavior of any API.

Server-only Content Manager for iSeries APIs

The following Content Manager for iSeries API exists on the server only; there is no API of a similar name on the workstation.

QVISNDRCV (Send and Receive Buffer)

Purpose

QVISNDRV is a generic function for sending data to and receiving data from a workstation. This function can be used by Content Manager for iSeries applications to display documents through the Content Manager for iSeries client. A reset option is also included to close the document workstation.

Parameters

Communication_Type

INT—input/output

The communication type to use. Valid values are:

0 Detect

The connection used for the application will be used, as determined by the device description. Value will be returned as 1

or 2, unless an error occurs. This would be used except when a specific workstation address is to be used, such as for printing.

- 1 APC (CPI-C). For explicitly using APPC.
- 2 TPC/IP. For example using TCP/IP.

Partner_Address

CHAR[20]—input/output

Address for the workstation with at least one trailing blank. This may be the fully qualified LU name for CPI-C or the TCP/IP address. If *Communication_Type* is set to 0, this field is ignored, but the workstation address will be returned here.

Partner_TPName

CHAR[20] — input/output

Transaction program name for APPC. If passed as blank, the default is EKDVICLA, which is provided by Content Manager for iSeries.

Partner_ModeName

CHAR[10]

Mode name for APPC, with at least one trailing blank. If passed as blank, mode name will be #INTER.

Partner_PortNumber

INT — input/output

For a TCP/IP connection, the port number on the workstation. If passed as 0, the default is 31015.

communication_handle

CHAR[20]

Contains the communication handle. If blank and the buffer size is not zero, a conversation will be allocated or a socket will be opened to connect to the workstation. If the buffer size is 0, and this field is not blank, the conversation will be deallocated or the socket will be closed.

dllname

CHAR — input/output

The name of the DLL, null or blank terminated, to be loaded on the workstation. The function in the DLL must be:

```
int vi400comm (int * buffer_size, char * buffer)
```

If a non-zero return is received, the workstation program will be ended. The user would then have to start it again to be able to initiate another display request.

If passed as blank, the default is EKDVIDSP.DLL, which is provided by Content Manager for iSeries to support host-initiated display requests.

host_code_page

INT — input

If 0, QVISNDRCV will extract the current code page. All data in the buffer must be translatable characters. To send binary data that is not converted, use -1.

buffer_size

INT — input/output

Pointer to the size of the buffer to send from and receive into. The maximum size is 32760 bytes. If 0 on the host, the conversation or socket will be closed. If non-zero on return, the buffer contains data sent from the workstation. No more than 32500 bytes can be sent or received. The rest of the 32K is for control information.

buffer CHAR — input/output

Pointer to the buffer to send from and receive into. This must be at least as long as the *buffer_size* specified, or the size of the buffer returned. Providing a return buffer that is smaller than the amount of data returned will not cause an explicit error, but will probably cause the calling program to fail.

Return Values

The function returns an integer return code if an error occurs in the Content Manager for iSeries code.

Sample code is provided which supports host-initiated display requests using the Content Manager for iSeries. This code will return the following character return codes in the buffer passed back to the calling application:

- 1 Content Manager for iSeries was not started
- 2 Null buffer passed
- 3 First byte not R (reset) or D (display)
- 4 Invalid item ID length
- 5 Invalid item ID
- 6 Problem accessing item
- 7 Content Manager for iSeries error

Guidelines for Use

All parameters are passed by reference. Character variables may be null or blank terminated.

Create your custom modules using ILE C/400, ILE COBOL/400, ILE RPG/400, or VisualAge/400. Then create a program binding your new modules with service program QVISNDRCV.

Two workstation programs for communications are provided: EKDVICLA for APPC communications and EKDVICLT for TCP/IP communications. If called with defaults, the address of the workstation and the communication type will be determined automatically.

For APPC communications, the program EKDVICLA can be pre-started or defined as a transaction program to be started by the attach manager. If you are using Personal Communications for APPC support, to define the transaction program EKDVICLA, set *Receive_Allocate timeout* to 0, and check *Dynamically loaded*, *Queued TP*, and *Background process*. If the program is not already running when requested by a program on the iSeries, it will be automatically started. By setting the timeout to 0, the program will remain active even after the conversation is deallocated.

For TCP/IP communications, the program EKDVICLT must be pre-started on the workstation. If the port number (31015) is not acceptable, a different value may be passed as a parameter when starting EKDVICLT.

Sample Source

Refer to sample source program, QVIDSPTST, in file QCSRC in your QVI library. This program is provided as a sample for calling QVISNDRCV from a C program on the server. It contains, defines, and structures that you will find useful when creating your custom code.

Chapter 8. Content Manager for iSeries User Exits

User exits provided by Content Manager for iSeries are specific points in the program where you can specify your own processing routines. You may create exit programs which provide a level of customization by accessing a database or integrating with another application.

Client User Exits

The user exit points described here are invoked by the Content Manager for iSeries. Use the following user exits in conjunction with the Client for Windows.

AlternateSearchUserExit (alternate search user exit)

Format

```
SHORT AlternateSearchUserExit(hSession, hWnd, szUserID usTypeFilter,  
fWipFilter, usSuspendFilter, usIndexClass, usNumCriteria, pCriteria,  
pItemIdResultFolder)
```

Purpose

Use the **AlternateSearchUserExit** to replace the search function of the client application program with your own search routine. The exit returns the results of the search operation in a search result folder.

Parameters

hSession

HSESSION — input

Session handle returned by **SimLibLogon**.

hWnd HWND — input

The handle to a window. The device manager uses this handle to identify the window where any operation of an end-user interface occurs, such as the display of error messages.

pszUserID

PSZ — input

The 0-terminated character string containing the user ID of the user who receives the search results. This parameter is not case-sensitive.

usTypeFilter

USHORT — input

The type of items to search for. The valid values are:

SIM_DOCUMENT

Indicates that the item is a document.

SIM_FOLDER

Indicates that the item is a folder.

SIM_FOLDER_DOC

Indicates that the item can be either a folder or a document.

fWipFilter

BITS — input

The work-in-process status of the items to search for. The following are valid values. You can use a bit inclusive OR operator (|) to combine them.

OIM_ITEMS_NOT_IN_WORKFLOW

Searches for items not in a workflow.

OIM_CURRENT_WORKFLOW_ITEMS

Searches for items in a workflow.

OIM_CANCELLED_WORKFLOW_ITEMS

Searches for items removed from a workflow.

OIM_COMPLETED_WORKFLOW_ITEMS

Searches for items that completed their workflow.

OIM_ALL

Searches without regard for the work-in-process status of the object. Do not combine this value with the others. It is equivalent to using all the other values.

usSuspendFilter

USHORT — input

The suspension status of the items to search for. The valid values are:

OIM_ITEMS_SUSPENDED

Searches for suspended items.

OIM_ITEMS_NOT_SUSPENDED

Searches for items that are not suspended.

OIM_ALL

Searches without regard for the suspension status of the object. Do not combine this value with the others. It is equivalent to using all the other values.

usIndexClass

USHORT — input

The index class identifier of the index class for the folder you create for the search results. Ensure that the index class you assign to the created folder has no required attributes. Otherwise, the search fails and the folder is not created.

If you do not want to assign an index class to the folder you create, specify the value 0 for this parameter.

If the value of the *fMemListRequest* parameter is TRUE or the value of the *usStatDyn* parameter is SIM_SEARCH_BUILD_ONLY, IBM Content Manager for iSeries ignores this value.

usNumCriteria

USHORT — input

The number of elements in the *pCriteria* array.

pCriteria

PLIBSEARCHCRITERIASTRUCT — input

The pointer to an array specifying the search criteria for each view to be searched. The array it points to must have at least one element.

pItemIdResultFolder

PITEMID — output

The pointer to the search results folder.

Return values

The exit returns SIM_RC_OK to indicate that the search operation completed normally. All other return values indicate an abnormal ending and are logged as errors.

On successful completion, the function identifies the search results folder in the value of the *ItemIdResultFolder* output parameter.

Comments

The Alternate Search user exit routine works at the view level. When running a basic search, if the search is against a particular view, the client application program loads the exit for that view. If the search is against all views, the client application program loads the exit for the base view of the NOINDEX class. For advanced search, the client application program loads the exit for the base view of the NOINDEX class.

ChangeSMSUserExit (change system-managed storage user exit)

Format

```
SHORT ChangeSMSUserExit(hwnd, pExitStruct, pfContinue)
```

Purpose

This user exit routine is called whenever the index class is changed for an item before the library object window is closed. The exit is passed the ItemID of the item and returns a flag indicating whether default processing should continue. The default processing calls **SimLibChangeObjSMS** for each of the item's parts using the object server and collection information defined in the item's new index class.

Use the system administration program to specify this user exit routine in the settings notebook of the index class. Refer to the *System Administration Guide*.

Parameters

hwnd HWND — Input

Anchor window for message boxes. This parameter can be used to display messages and associate them with the application window.

pExitStruct

PUSEREXITSTRUCT — Input

User-defined attribute fields and other relevant information for the open document are passed in the *pExitStruct* parameter.

pfContinue

PBOOL — Output

Pointer to the continue flag. Set this value to TRUE to continue with default processing.

Internal representation

USEREXITSTRUCT:

```
typedef struct
{
    HSESSION
        hSession;
    ITEMID
        uidItem;
    USHORT
        itemidWorkflowId;
    BOOL flsUnindexed;
    USHORT
        hOrigClass;
    USHORT
        hClass;
    CHAR
        szUserId[LST_USERID_LEN+1];
    CHAR
        szUserHandle[LST_USERID_LEN+1];
    USHORT
        usAccessLevel;
    SHORT
        sFields;
    FIELDVALUE *
        pFields;
} USEREXITSTRUCT;

typedef USEREXITSTRUCT * PUSEREXITSTRUCT
```

where:

hSession

Session handle returned by **SimLibLogon**.

uidItem

Is the ItemID of the current document or folder to be changed.

itemidWorkflowId

Is the workflow ID of the opened document or folder to be changed. This value is NULL if the object is not in a workflow.

flsUnindexed

This value is TRUE if the object is a new document that has not been indexed in the system.

hOrigClass

Is the original class ID of the opened document or folder.

hClass

Is the current class ID of the opened document or folder.

szUserId[LST_USERID_LEN+1]

Is the user ID of the user saving the document or folder.

szUserHandle[LST_USERID_LEN+1]

This parameter is reserved.

usAccessLevel

Is the access privilege the user has for this document or folder. The valid value is:

UX_PRIV_WRITE when the user opens this object in UPDATE mode.

sFields Is the number of fields passed to the exit in the *pFields* parameter.

pFields Is the pointer to an array of FIELDVALUE data structures. The configuration and content of the user-defined attributes for this document or folder is passed to the exit in these data structures.

FIELDVALUE:

```
typedef struct
{
    USHORT
        usFieldId;
    USHORT
        usDataType;
    USHORT
        usMaxLength;
    BOOL flsReq;
    PSZ pBuffer;
} FIELDVALUE;
```

```
typedef FIELDVALUE * PFIELDVALUE
```

where:

usFieldId

Is the user-defined attribute ID.

usDataType

Is IBM Content Manager for iSeries data type of the attribute in the *usFieldId* parameter. This is a numeric equivalent representing the data type.

usMaxLength

Is the maximum number of bytes in the *pBuffer* parameter to appear in the Index Form window, excluding the NULL terminator.

flsReq This value is TRUE if the field is required.

pBuffer

Is the current value of the attribute in ASCIIZ display format. The buffer length is the value in the *usMaxLength* parameter plus one for the NULL terminator.

Results

The function returns SHORT with zero as SUCCESS. If any value other than zero is returned, default processing occurs.

If the call is successful, the value returned in the *pfContinue* parameter is checked.

Comments

The exit routine must not free the buffers that are passed in. All items sent to the exit are read-only copies. This exit must not modify these data structures.

The index form is closing when this user exit routine is called.

If a class has both the Save Record and Change SMS user exit routines specified, the Save Record user exit routine is called first.

DetNextWBUserExit (determine next workbasket user exit)

Format

```
SHORT DetNextWBUserExit( hwnd, usOperation, sNumberofITEMIDs,  
pListofITEMIDs, pExitStruct, pNextWorkBasketITEMID, pfComplete, pfContinue)
```

Purpose

The client application program calls this user exit routine from one of three functions within IBM Content Manager for iSeries. The exit is associated with a particular index class. This exit routes an item of this class to another workbasket, starts the item in a workflow, or changes its workflow.

The client application program calls this user exit routine whenever the user chooses the Route to option on the Process menu if the index class of the item defines the exit. By default, IBM Content Manager for iSeries determines if the item is in a workflow. If it is, it determines the next workbasket in the workflow. The system selects this workbasket in the resulting dialog box.

The client application program calls this user exit routine prior to displaying the Route To dialog box, regardless of whether the item is in a workflow. If the user exit routine returns a workbasket ITEMID, the workbasket appears as selected in the Route To dialog box. The user can still select a different workbasket in which to route the items. The user exit routine can perform any required processing and notifies IBM Content Manager for iSeries that the route operation should not continue. In this case, the Route To dialog box does not appear.

The client application also calls this user exit routine when the user selects the Start workflow or Change workflow option on the Process menu. The default processing for the Start workflow option includes routing the item to the first workbasket in the workflow. For a Change workflow action, the user can optionally route the item to the first workbasket.

The system does not call the Determine Next Workbasket User Exit during an automatic workflow operation.

The client application program calls this user exit routine prior to the actual routing of the item. The system routes this item to the specified workbasket. A valid workbasket must be returned in this case, because an item in a workflow must always be in a workbasket, even if the workbasket is not part of the workflow.

Use the system administration program to specify this user exit routine in the Next workbasket field of the index class settings notebook. Refer to the *IBM Content Manager for iSeries: System Administration Guide*.

Parameters

hwnd HWND — input

Anchor window for message boxes. You can use this parameter to display messages and associate them with the application window.

usOperation

USHORT — input

This value indicates the operation that called the user exit routine. The value is one of the following:

- UX_ROUTE
- UX_START_WORKFLOW
- UX_CHANGE_WORKFLOW

sNumberOfITEMIDs

USHORT — input

This value specifies the number of ItemIDs in the list that is pointed to by the *pListofITEMIDs* parameter. If this number is greater than one, the user selects multiple objects from the table of contents of a folder or workbasket.

pListofITEMIDs

PITEMID — input

This parameter is the pointer to the list of ItemIDs for the documents and folders the user wants to route.

pExitStruct

PUSEREXITSTRUCT — input

If the document or folder being routed is open when the exit is called, the user-defined attributes for the object and other relevant information are passed in the *pExitStruct* parameter. The values in the data structure include changes made to the class and attributes in the Index Form window.

If the object being routed is not open, the *pListofITEMIDs* parameter points to a list of one or more ItemIDs the user selects from the Table of Contents window. The *pExitStruct* values are NULL except for the *szUserId* parameter, that contains the current value.

pNextWorkBasketITEMID

PITEMID — input/output

Initially contains a pointer to the ItemID of the next workbasket recommended by IBM Content Manager for iSeries. If the document or folder being routed is not in a workflow, the initial ItemID value contains zeros. Replace this value only if the user exit routine returns the ItemID of a valid workbasket in the system.

pfComplete

PBOOL — input/output

Set this parameter to TRUE if IBM Content Manager for iSeries recommends marking this object as complete for the workflow when the user exit routine returns control. When the document or folder is marked complete, the system automatically removes it from the workbasket.

Recommendation: Do not set this parameter to TRUE if a user selects multiple workflow objects that should not be marked as complete.

pfContinue

PBOOL — output

Set this parameter to FALSE to cancel the route to action. This value lets the user exit routine perform all routing without letting the user override the suggestion. If you set this flag to FALSE, the Route To dialog box does not appear. The system ignores this parameter when the user exit routine is called during a Save, Start workflow, or Change workflow operation.

Internal representation

USEREXITSTRUCT::

```
typedef struct
{
    HSESSION
        hSession;
    ITEMID
        uidItem;
    ITEMID
        itemidWorkflowid;
    BOOL flsUnindexed;
    USHORT
        hOrigClass;
    USHORT
        hClass;
    CHAR
        szUserId[LST_USERID_LEN+1];
    CHAR
        szUserHandle[LST_USERID_LEN+1];
    USHORT
        usAccessLevel;
    SHORT
        sFields;
    FIELDVALUE *
        pFields;
} USEREXITSTRUCT;

typedef USEREXITSTRUCT * PUSEREXITSTRUCT
```

where:

uidItem

Is the ItemID of the current document or folder the user wants to route if only one item is being routed. If the user wants to route more than one item, the value is NULL. This value is NULL if the user does not open this object.

itemidWorkflowId

If called during a Start workflow action or Change workflow action, this value is the workflow ID of the document or folder the user wants to route. The value is also the workflow ID if the user selects the route to action for a single document or folder. If the user selects the Route to option for more than one document or folder, the value is NULL.

flsUnindexed

This value is always NULL.

hOrigClass

This value is always NULL.

hClass This value is always NULL.

szUserId[LST_USERID_LEN+1]

This value is the user ID of the user routing the document or folder.

szUserHandle[LST_USERID_LEN+1]

This parameter is reserved.

usAccessLevel

This value is always NULL.

sFields This value is always NULL.

pFields This value is always NULL.

FIELDVALUE:

```
typedef struct
{
    USHORT
        usFieldId;
    USHORT
        usDataType;
    USHORT
        usMaxLength;
    BOOL flsReq;
    PSZ pBuffer;
} FIELDVALUE;
```

```
typedef FIELDVALUE * PFIELDVALUE
```

where:

usFieldId

Is the user-defined attribute identifier.

usDataType

Is IBM Content Manager for iSeries data type of the attribute in the *usFieldId* parameter. This is a numeric equivalent representing the data type.

usMaxLength

Is the maximum number of bytes in the *pBuffer* parameter to appear in the Index Form window, excluding the NULL terminator.

flsReq This value is TRUE if the field is required.

pBuffer

Is the current value of the attribute in ASCIIZ display format. The buffer length is the value in the *usMaxLength* parameter plus one to represent the NULL terminator.

Results

The function returns a value of SHORT with zero for SUCCESS. Another value is assumed to be an error and an error message appears.

If the exit completes successfully, the *pfComplete* parameter is checked. If this parameter is set to TRUE, IBM Content Manager for iSeries displays a message box that recommends marking the selected objects as complete for this workflow. This parameter is ignored if the object is not in a workflow.

If the *pfComplete* parameter is not set to TRUE, the value in the *pNextWorkBasketITEMID* parameter is used as the recommended destination for the selected objects. This value must point to a valid IBM Content Manager for iSeries workbasket ItemID. The user can override these recommendations for both cases, either the next workbasket or completion. If the user exit routine is called during a route operation and *pfContinue* is FALSE, the Route To dialog box does not appear.

Comments

The exit routine must not free the buffers that are passed in. All items sent to the exit are read-only copies. These data structures must not be modified by this exit. Do not perform any OIM function calls to change the workflow status or

workbasket of an object listed in the data structures or parameters of this user exit routine unless the *pfContinue* parameter is set to FALSE.

When the user exit routine is called during a Route to option from the Process menu, the parameters in the USEREXITSTRUCT data structure are NULL except for the *szUserId* parameter. When the user exit routine is called during a Start workflow or Change workflow action from the Process menu, the parameters in the USEREXITSTRUCT data structure are NULL except for the *itemidWorkflowId* and *szUserId* parameters. In these cases, the FIELDVALUE data structures that normally contain details about the user-defined attributes are not passed to the user exit routine. If you need information about the user-defined attributes to process this exit, use the appropriate OIM function calls to obtain the required data. Refer to the following function calls:

- SimLibGetAttrInfo
- SimLibGetClassInfo
- SimLibGetItemInfo
- SimLibGetItemType
- SimLibItemSnapshot
- SimLibOpenItem
- SimLibReadItemAttr

When multiple items are selected from the table of contents and routed, the class of the first item in the list is checked for the DetNextWBUserExit. If this user exit routine is specified for the first class, this user exit routine is called for all items selected, regardless of their classes. If the first item's class does not have a user exit routine specified, no user exit routines are called.

DetermineWorkflowUserExit (determine workflow user exit)

Format

```
SHORT DetWorkflowUserExit( hwnd, puidItem, pExitStruct, puidWorkflow, puidWorkbasket)
```

Purpose

The client application program calls this user exit routine when a user saves a document or folder with an index class that is defined to automatically start items in a workflow when they are saved. The client application program calls this user exit routine only when these items have never been in a workflow before.

Although this user exit routine is specified for a particular index class, the same user exit routine can be used for multiple index classes.

IBM Content Manager for iSeries automatically provides the user exit routine with the default workflow for the index class, as specified by the system administrator. The user exit routine can specify that the item should be started in a different workflow, the default workflow, or no workflow.

This user exit routine can also optionally specify the workbasket where the item is to be routed. When an item is specified to be in a workflow, it must be in a workbasket even if the workbasket is not in the workflow where the item is. If the user exit routine does not explicitly specify a workbasket, IBM Content Manager for iSeries routes the item to the first workbasket in the workflow.

Use the system administration program to specify this user exit routine in the Automatic workflow field of the index class settings notebook. Refer to the *Administration and Operation Guide*.

Parameters

hwnd HWND — Input

Anchor window for message boxes. This parameter can be used to display messages and associate them with the application window.

puidItem

ITEMID — input

Pointer to the ItemID of the item being saved.

pExitStruct

PUSEREXITSTRUCT — Input

User-defined attributes for the document or folder and other relevant information are passed in the *pExitStruct* parameter.

puidWorkflow

ITEM — Input/Output

Pointer to the workflow item ID that the item should be started in. The workflow ID provided as input to the user exit routine is the default workflow for the class, as defined by the system administrator.

The user exit routine should set the workflow ID to one of the following:

no change

Uses the default workflow.

workflow item id

Where the item is to be started must be defined.

a null id (Set at least the first character of UID null)

Cancels automatic workflow processing. The item is not started in a workflow.

puidWorkBasket

ITEMID — Output

Pointer to the Workbasket ITEMID that the item being saved should be routed to after it is started in the workflow. This parameter points to a NULL ITEMID when the exit is called. The user exit routine should set this parameter to a valid workbasket ItemID if the item should be routed to a workbasket other than the first workbasket in the workflow. If this parameter is still a NULL ITEMID when the user exit returns, IBM Content Manager for iSeries routes the item to the first workbasket in the workflow.

Internal representation

USEREXITSTRUCT:

```
typedef struct
{
    HSESSION
        hSession
    ITEMID
        uidItem;
    USHORT
        itemidWorkflowId;
    BOOL flsUnindexed;
    USHORT
        hOrigClass;
}
```

```

USHORT
    hClass;
CHAR
    szUserId[LST_USERID_LEN+1];
CHAR
    szUserHandle[LST_USERID_LEN+1];
USHORT
    usAccessLevel;
SHORT
    sFields;
FIELDVALUE *
    pFields;
} USEREXITSTRUCT;

```

```
typedef USEREXITSTRUCT * PUSEREXITSTRUCT
```

where:

hSession

Session handle returned by **SimLibLogon**.

uidItem

Is the ItemID of the current document or folder to be saved.

itemidWorkflowId

This parameter is always null.

flsUnindexed

This value is TRUE if the object is a new document that has not been indexed in the system.

hOrigClass

Is the original class ID of the opened document or folder.

hClass

Is the current class ID of the opened document or folder. This value is the same as the *hOrigClass* parameter unless the user specifies a new index class.

szUserId[LST_USERID_LEN+1]

Is the user ID of the user saving the document or folder.

szUserHandle[LST_USERID_LEN+1]

This parameter is reserved.

usAccessLevel

Is the access privilege the user has for this document or folder. The valid value for this user exit is:

UX_PRIV_WRITE when the user opens this object in UPDATE mode.

sFields

Is the number of fields passed to the exit in the *pFields* parameter.

pFields

Is the pointer to an array of FIELDVALUE data structures. The configuration and content of the user-defined attributes for this document or folder are passed to the exit in these data structures.

FIELDVALUE:

```

typedef struct
{
    USHORT
        usFieldId;
    USHORT
        usDataType;
}

```


USHORT

```
        usMaxLength;  
    BOOL flsReq;  
    PSZ  pBuffer;  
} FIELDVALUE;
```

```
typedef FIELDVALUE * PFIELDVALUE
```

where:

usFieldId

Is the user-defined attribute ID.

usDataType

Is IBM Content Manager for iSeries data type of the attribute in the *usFieldId* parameter. This is a numeric equivalent representing the data type. Refer to the section "Attribute types" in the frnphi.h header file for the define statements and content requirements for these numbers.

usMaxLength

Is the maximum number of bytes in the *pBuffer* parameter to appear in the Index Form window, excluding the NULL terminator.

flsReq This value is TRUE if the field is required. If this parameter is set to TRUE and this FIELDVALUE data structure is modified by the exit, the value in *pBuffer* must not be changed to NULL.

pBuffer Is the current value of the attribute in ASCIIZ display format. The buffer length is the value in the *usMaxLength* parameter plus one to represent the NULL terminator.

Results

This user exit routine returns a value of type SHORT. It should return a value of zero for successful completion of the user exit routine. If it returns another value, the item is not started in a workflow, and an error message appears.

If the user exit routine completes successfully, the item is started in the workflow specified by the *puidWorkflow* parameter. If this parameter specifies a workflow ID of null, then the item is not started in a workflow and the automatic workflow processing is canceled. If the user exit routine specified a workbasket to route the item to, the item is routed to that workbasket by IBM Content Manager for iSeries after the item is started in the workflow. If the user exit does not specify a workbasket, the item is routed to the first workbasket in the workflow.

Comments

This user exit routine must not modify or free any of the buffers that are passed in. Do not start this item in a workflow with the **Ip2StartWorkFlow** function in this user exit routine unless the exit cancels workflow processing by returning null for the workflow ID.

This exit is called when a document or folder is saved after modifying the index values or changing the class of the item. It is not called if index values are not modified or if the Index Form is not open when the item is saved. This exit is called after the Save Record user exit routine is called.

Automatic workflow processing is performed only if the item being saved is not in a workflow and has never been in a workflow. Therefore, this user exit routine is called only if the item being saved has never been in a workflow.

This user exit routine is called when folders with an index class that specifies this user exit routine are created during auto-filing.

The index values that are passed to the user exit routine are in display format, not in the internal format in which data is stored in the database.

GetAttributeValueList (Get attribute value list)

Format

```
INT_cdecl GetAttributeValueList(hSession, nClassView, nAttrID, reserved,  
pControlType, pSortOption, ListValues, pNumValues, nMaxValueLen)
```

Purpose

This user exit routine allows you to extend the Edit Index window to include a combination list box. The actual values to be listed are returned by this user exit routine. This user exit routine must be contained in a Dynamic Library Link (DLL) named `frnwueal.dll`. This user exit routine is called for each attribute ID in an index class. The routine returns:

- Information about the type of control (entry field, combination list box with an entry field, combination list box without an entry field).
- The ordering option: whether to sort the list or display it as listed in the array.
- The list of values to display.

A sample of this user exit is located in the `%FRNROOT%\SAMPLES` directory.

Parameters

hSession

HSESSION - Input

Session handle returned by SimLibLogon.

nClassView

INT - Input

The index class view for which the *nAttrID* field, below, is being checked to determine the desired control type.

nAttrID

INT - Input

The key field being checked for the desired control type.

reserved

PVOID

Reserved parameter for future use; currently set to NULL.

pControlType

INT * - Output

Returns one of the following:

- 0 for a standard entry field
- 1 for a list box that allows text entry
- 2 for a static list box

pSortOption

INT * - Output

Returns one of the following:

- 0 to leave the combo values in the order returned
- 1 to have the list sorted alphabetically

ListValues

PPSZ - Output

An array of character pointers, each pointing to a zero-terminated string representing one of the values that will be displayed in the list box. If the field is a standard edit field (*pControlType=0), this array should have the size of 1. This also means that GetValueListLength, below, should return 1 in *pNumValues.

Restriction: Do not fill in more values than are specified in pNumValues, below.

pNumValues

INT * - Input and output

On input, this is the number of values returned in *pNumValues* from the GetValueListLength() function. This value can be left alone or decreased if fewer values are actually used. This value must not be incremented.

nMaxValueLen

INT - Input

The value that is returned through *pMaxValueLen* in GetValueListLength().

Results/Return Values

0 for success; non-zero for error.

Comments

Because this user exit routine is called for every field in the Edit Index window, it must run quickly.

GetValueListLength (Get value list length)

Format

```
INT_cdecl GetValueListLength(hSession, nClassView, nAttrID, reserved,
pNumValues, pMaxValueLen)
```

Purpose

This user exit routine returns the number of values and the maximum value length for the specified attribute ID, from the specified index class. The client calls this function for every field of every index class to determine if there is a list of values for the attribute and if there is, how much space to allocate for it. This user exit routine must be contained in a DLL named frnwuea1.dll.

A sample of this user exit routine is located in the %FRNROOT%\SAMPLES directory.

Parameters

hSession

HSESSION - Input

Session handle returned by SimLibLogon.

nClassView

INT - Input

The index class view for which the *nAttrID* field, below, is being checked to determine the desired control type.

nAttrID

INT - Input

The attribute being checked for the desired control type.

pNumValues

INT * - Output

The default is 0 (*pNumValues=0). If the attribute is to have values, set this to the number of values.

pMaxValueLen

INT * - Output

The default is 0 (*pMaxValueLen=0). If the attribute is to have values, set this to the maximum length of any value.

Results/Return Values

0 for success; non-zero for error.

Comments

This function gets called for every field in the Edit Index dialog so be sure that it works quickly.

OverloadTriggerUserExit (overload trigger user exit)

Format

```
SHORT OverloadTriggerUserExit(hwnd, usOperation, usNumberofITEMIDs, usIndex, pListofITEMIDs, pExitStruct, pWorkBasketITEMID, pNewWorkBasketITEMID)
```

Purpose

This user exit routine is called every time a document or folder is added to a workbasket that has reached its overload condition, except when added as a result of satisfying suspension criteria. The user adds an item to a workbasket by selecting the Route to option from the Process menu, then by selecting a destination from the list of available workbaskets in the system. Items are also added to a workbasket when a new class is specified that automatically assigns the item to a workflow. This user exit routine can also be called during a Start workflow or Change workflow operation, or during a scanning or importing operation.

The suspension criteria include:

- A timeout as detected by the expired time check utility.
- Adding an item to a folder using the **SimLibAddFolderItem** function. Adding an item to a folder through the user interface triggers this user exit routine.

The overload trigger is the number specified in the system administration program for the maximum quantity of items allowed in the workbasket. If the overload condition is triggered for the workbasket, the user exit routine is processed.

By default, IBM Content Manager for iSeries displays a message that the overload condition has occurred, and lets the user cancel the route, select a different workbasket as the destination, or force the items into the original workbasket. This

user exit routine can also be used to replace the default IBM Content Manager for iSeries processing. You can specify an alternate workbasket to be used as the backup and return the ITEMID of that alternate to IBM Content Manager for iSeries.

Use the system administration program to specify this user exit routine in the workbasket settings notebook. Refer to the *Administration and Operation Guide* .

Parameters

hwnd HWND — Input

Anchor window for message boxes. This parameter can be used to display messages and associate them with the application window.

usOperation

USHORT — input

This value indicates the operation that called the user exit routine. The value is one of the following:

- UX_SAVE_ITEM
- UX_ROUTE
- UX_START_WORKFLOW
- UX_CHANGE_WORKFLOW
- UX_SCAN_ITEM
- UX_IMPORT_ITEM

usNumberofITEMIDs

USHORT — Input

Number of ItemIDs in the *pListofITEMIDs* parameter.

usIndex

USHORT — Input

The item that cause the overload condition to occur.

pListofITEMIDs

PITEMID — Input

Pointer to a list of ItemIDs of the items to be routed to the workbasket.

pExitStruct

PUSEREXITSTRUCT — Input/output

If the document or folder being routed is open at exit processing time, the user-defined attribute fields and other relevant information for the object are passed in the *pExitStruct* parameter. The values in the data structure include changes the user made to the class and attributes in the Index Form window.

If the object being routed is not open, the *pListofITEMIDs* parameter points to a list of one or more ItemIDS selected by the user from the Table of Contents window. The *pExitStruct* parameters are NULL except for *szUserId*, that contains the current value.

pWorkBasketITEMID

PITEMID — Input

Pointer to the ITEMID of the original destination workbasket causing the overload trigger.

pNewWorkBasketITEMID

PITEMID — Output

Pointer to a buffer containing a NULL ITEMID. Replace this with the ItemID of a valid workbasket in the system to be used as a backup destination.

Internal representation

USEREXITSTRUCT:

```
typedef struct
{
    HSESSION
        hSession
    ITEMID
        uidItem;
    USHORT
        itemidWorkflowId;
    BOOL flsUnindexed;
    USHORT
        hOrigClass;
    USHORT
        hClass;
    CHAR
        szUserId[LST_USERID_LEN+1];
    CHAR
        szUserHandle[LST_USERID_LEN+1];
    USHORT
        usAccessLevel;
    SHORT
        sFields;
    FIELDVALUE *
        pFields;
} USEREXITSTRUCT;
```

```
typedef USEREXITSTRUCT * PUSEREXITSTRUCT
```

where:

hSession

Session handle returned by **SimLibLogon**.

uidItem

Is the ItemID of the current document or folder to be routed.

itemidWorkflowId

Is the workflow ID of the opened document or folder to be routed. This value is NULL if the object is not opened by this user or if the object is not in a workflow.

flsUnindexed

This value is TRUE if the object is a new document that has not been indexed in the system. This value is FALSE if the object is not opened by this user.

hOrigClass

Is the original class ID of the opened document or folder. This value is NULL if the object is not opened by this user.

hClass

Is the current class ID of the opened document or folder. This value is the same as the *hOrigClass* parameter unless the user specified a new index class. This value is NULL if the object is not opened by this user.

szUserId[LST_USERID_LEN+1]

Is the user ID of the user routing the document or folder.

szUserHandle[LST_USERID_LEN+1]

This parameter is reserved.

usAccessLevel

Is the access privileges the user has for this document or folder. This value is NULL if the object is not opened by this user. The valid values are:

UX_PRIV_READ when the user opens this object in BROWSE mode.

UX_PRIV_WRITE when the user opens this object in UPDATE mode.

sFields Is the number of fields passed to the exit in the *pFields* parameter. This value is zero if the object is not opened by this user or if the user selects the Route to option for an opened document or folder while the Index Form window for that object is closed.

pFields Is the pointer to an array of FIELDVALUE data structures. The configuration and content of the user-defined attributes for this document or folder are passed to the exit in these data structures. This value is NULL if the object is not opened by this user or if the user selects the Route to option for an opened document or folder while the Index Form window for that object is closed.

FIELDVALUE:

```
typedef struct
{
    USHORT
        usFieldId;
    USHORT
        usDataType;
    USHORT
        usMaxLength;
    BOOL flsReq;
    PSZ pBuffer;
} FIELDVALUE;
```

```
typedef FIELDVALUE * PFIELDVALUE
```

where:

usFieldId

Is the user-defined attribute ID.

usDataType

Is IBM Content Manager for iSeries data type of the attribute in the *usFieldId* parameter. This is a numeric equivalent representing the data type. Refer to the section "Attribute types" in the frnpfi.h header file for the define statements and content requirements for these numbers.

usMaxLength

Is the maximum number of bytes in the *pBuffer* parameter to appear in the Index Form window, excluding the NULL terminator.

flsReq This value is TRUE if the field is required.

pBuffer Is the current value of the attribute in ASCIIZ display format. The buffer length is the value in the *usMaxLength* parameter plus one to represent the NULL terminator.

Results

The function returns a value of `SHORT` with zero as `SUCCESS`. If the call completes successfully, the value in the `pNewWorkBasketITEMID` parameter is used as the alternate destination. This must be a valid IBM Content Manager for iSeries workbasket ItemID. If this ItemID is the same as the overloaded workbasket or if this value contains a `NULL` ITEMID, the items are forced into the original workbasket and the overload condition is ignored.

If the call returns any value other than zero, an error message appears, and the items are not routed to any workbasket.

If an error is returned while the user exit routine is called during a save, the item is saved but not placed in a workflow or routed to any workbasket.

If the new workbasket routing results in another overload, this user exit routine is called again.

Comments

The exit routine must not free the buffers that are passed in. All parameters sent to the exit are read-only copies. Buffers should not be modified or unallocated.

If the `pNewWorkBasketITEMID` parameter is still `NULL` after the exit completes, the selected items are forced into the original workbasket destination.

If the Index Form window is not opened when the user selects the Route to option from the Process menu, the `pFields` pointer and the `sFields` parameter in the `USEREXITSTRUCT` data structure are `NULL`. In this case, the `FIELDVALUE` data structures that normally contain details about the user-defined attributes are not passed to the user exit routine.

Avoid assigning two workbaskets as backup for each other. In this case, you can begin an endless loop of circular references if both workbaskets are overloaded.

QuerySortUserExit (query sort user exit)

Format

```
SHORT QuerySortUserExit( hSession, hwnd, pSortList, usItemCount, pszUserId, usSortObject)
```

Purpose

This user exit routine is called when a folder or workbasket is opened that contains documents or folders in a class for which the exit is defined. This includes a folder created as a result of a filerom search. You can program this exit to sort and modify the table of contents of the folder or workbasket before it appears on the screen. This function lets you define a specific sort order other than the default ascending or descending order provided by IBM Content Manager for iSeries. The exit can also be used to filter out selected documents and folders to prevent display and user access to those objects. It can also be called prior to printing the table of contents of a folder.

You can assign this user exit routine on a class basis using IBM Content Manager for iSeries. Each class represented in a folder or workbasket table of contents is sorted according to the user exit routine specified for that class. If more than one class in the folder or workbasket calls this user exit routine, each exit routine is

called and completed sequentially prior to the display of the contents. Only the documents and folders assigned to a specific class are passed to the exit routine called for that class.

Use the system administration program to specify this user exit routine in the Sort field of the index class settings notebook. Refer to the *Administration and Operation Guide*.

Parameters

hSession

HSESSION — Input

Session handle returned by **SimLibLogon**.

hwnd HWND — Input

Anchor window for message boxes. This parameter can be used to display messages and associate them with the application window.

pSortList

USERSORTSTRUCT — Input/Output

Pointer to an array of documents and folders to be sorted. Each document or folder is represented by a USERSORTSTRUCT data structure.

usItemCount

USHORT — Input

Number of documents and folders in the *pSortList* parameter.

pszUserId

PSZ — Input

User ID name of the user opening the folder or workbasket. This is the ID specified through the API.

usSortObject

USHORT — Input

Type of object that appears. The valid values are:

SIM_FOLDER when the table of contents is sorted for a folder display. This includes search results folders.

SIM_WORKBASKET when the table of contents is sorted for a workbasket display.

Internal representation

USERSORTSTRUCT:

```
typedef struct
{
    USHORT
        usType;
    USHORT
        usClass;
    ITEMID
        uid;
    USHORT
        usPriority;
    PCHAR *
        pszVals;
    PCHAR *
        pszWbVals;
```

```

ATLIST *
    pAttrList;
BOOL fCheckedOut;
USHORT
    usFlags;
} USERSORTSTRUCT;

```

```
typedef USERSORTSTRUCT * PUSERSORTSTRUCT;
```

where:

usType Is the type of object. The valid values are:
 SIM_DOCUMENT when the object is a document.
 SIM_FOLDER when the object is a folder.

usClass
 Is the current view identifier of the index class for this object.

uid Is the IBM Content Manager for iSeries ITEMID of this object.

usPriority
 Is the priority for this object.

pszVals
 Is the pointer to an array of display values in ASCIIZ format for this document or folder. These values include the user-defined attributes and the following system attributes:
 Workflow name
 Priority
 Check-out ID
 Suspend status.

A NULL ASCIIZ string appears in the array for each attribute that is not in the user's current layout for this index class. For each value in this array, there is a corresponding value in the ATLIST data structure from the *pAttrList* parameter.

pszWbVals
 Is the pointer to the workbasket view values for an object. The values included are in this order:
 Priority
 Date of entry to the workbasket
 Time of entry to the workbasket
 Class name

This is the time and date stamp indicating the time of entry to this workbasket. This pointer is NULL if the *usSortObject* is equal to SIM_FOLDER.

pAttrList
 Is the pointer to an ATLIST data structure. This data structure contains detailed information about the attribute values that appear for this object. These values include the user-defined attributes and the following system attributes:
 Workflow name
 Priority
 Check-out ID
 Suspend status.

fCheckedOut
 This value is TRUE if the object is checked out.

usFlags

Set this parameter to SF_HIDE if this object should not appear in the sorted table of contents. The count reflects items that are not hidden.

ATLIST:

```
typedef struct
{
    USHORT
        usClass;
    USHORT
        usCount;
    ATINFO *
        patinfo;
    USHORT
        usUserCount;
    USHORT *
        patidUserList;
} ATLIST;

typedef ATLIST * PATLIST;
```

where:

usClass

Is the current view id for the index class stored for this object. This is the same value as the *usClass* parameter in the USERSORTSTRUCT data structure.

usCount

Is the number of attributes listed in the array referred to in the *patinfo* parameter.

patinfo

Is the pointer to an array of ATINFO data structures. There is a separate data structure for each user-defined attribute and these system attributes:

- Workflow name
- Priority
- Check out ID
- Suspend status.

When this user exit routine is called from a print operation, only the attributes in the user's current layout are included in the array.

usUserCount

Is the number of attributes listed in the array referred to in the *patidUserList* parameter.

patidUserList

Is the pointer to an array of USHORTs. There is a separate USHORT for each user-defined attribute of this object to appear in the Table of Contents window. These attributes are selected by each user from the list of attributes assigned to this index class. Only these selected attributes can be viewed.

ATINFO:

```
typedef struct
{
    USHORT
        atid;
    PATTRINFOSTRUCT
        pai;
}
```

```
    } ATINFO;  
typedef ATINFO * PATINFO;
```

where:

- atid* Is the attribute ID defined in the ATTRINFOSTRUCT data structure pointed to by the *pai* parameter.
- pai* Is the pointer to an ATTRINFOSTRUCT data structure. This data structure contains the attribute name in the system, data type, minimum length, and maximum length.

Results

The function returns a value of SHORT with zero as SUCCESS. The table of contents of the folder or workbasket appears in random order.

If the exit completes successfully, the items appear in the order in which they are sorted in USERSORTSTRUCT array (*pSortList* [0], *pSortList* [1]). If the *usFlags* parameter is set to SF_HIDE, the document or folder does not appear with the other objects in its index class.

Comments

The exit routine must not free the buffers that are passed in. This exit does not allow changes to the user layout of the Table of Contents window.

The attribute values cannot be modified by the exit. These attributes are listed in the *patidUserList* parameter of the ATLIST data structures.

This exit is not called if the workbasket being opened is specified for system-assigned work through the system administration program. If the user displays a workbasket in priority mode, IBM Content Manager for iSeries ignores the order returned by the user exit routine. Items that are specified to be hidden do not appear.

This function is processed prior to the display of the list.

SaveRecordUserExit (save record user exit)

Format

```
SHORT SaveRecordUserExit( hwnd, pPreSaveStruct, ppszErrorMsgs,  
    ppusFieldIdsInError)
```

Purpose

This user exit routine is called when a user chooses to save changes to the user-defined attributes of a document or folder. The index attribute fields are passed to the exit for processing. The new attribute data entered in the Index Form window can be validated by matching the information in your existing files. This exit also allows changes to the user-defined attribute fields.

If the fields are modified by the exit, they are audited by IBM Content Manager for iSeries before the record is written to the database. The audits compare the data returned using the following guidelines:

- Data type — the format and content of the data must conform to the requirements of the data type for the attribute.

Minimum length — the minimum length requirement of the data string, or minimum numeric value for certain data types, must be met if specified for the attribute.

Maximum length — the maximum length requirement of the data string, or maximum numeric value for certain data types must be met if specified for the attribute.

Required fields must be specified.

The exit can return a list of error messages to indicate any errors in the user-specified values. The error messages appear in the Index Form Errors window. The display fields of the attributes corresponding to the error messages are flagged with a question mark in the Index Form window.

Use the system administration program to specify this user exit routine in the Save field in the index class settings notebook. Refer to the *Administration and Operation Guide*.

Parameters

hwnd HWND — Input

Anchor window for message boxes. This can be used to display messages and associate them with the application window.

Restriction Because the frame is disabled during the save, do not use this window (*hwnd*) as the parent of a dialog. You can use the desktop as the parent, and you can use this window (*hwnd*) as the owner.

pPreSaveStruct

USEREXITSTRUCT — Input/output

User-defined attributes for the document or folder and other relevant information are passed in the *pPreSaveStruct* parameter.

ppszErrorMsgs

PSZ * — Output

Address of a pointer. The pointer must be set by your exit routine to point to a data stream of ASCII strings representing error messages. Each error message must correspond with an attribute ID in the *ppusFieldIdsInError* parameter. The required format of the error data stream is defined in the Results section. This buffer must be allocated by the user exit routine and is deallocated by IBM Content Manager for iSeries.

ppusFieldIdsInError

PUSHORT * — Output

Address of a pointer to an array of attribute IDs associated with the error messages returned in the *ppszErrorMsgs* parameter. The valid attribute IDs are passed to the exit in the *usFieldId* parameter of the FIELDVALUE data structures. This buffer must be allocated by the user exit routine; it is deallocated by IBM Content Manager for iSeries.

Internal representation

USEREXITSTRUCT:

```
typedef struct
{
    HSESSION
    hSession
```

```

ITEMID
    uidItem;
USHORT
    itemidWorkflowId;
BOOL flsUnindexed;
USHORT
    hOrigClass;
USHORT
    hClass;
CHAR
    szUserId[LST_USERID_LEN+1];
CHAR
    szUserHandle[LST_USERID_LEN+1];
USHORT
    usAccessLevel;
SHORT
    sFields;
FIELDVALUE *
    pFields;
} USEREXITSTRUCT;

```

```
typedef USEREXITSTRUCT * PUSEREXITSTRUCT
```

where:

hSession

Session handle returned by **SimLibLogon**.

uidItem

Is the ItemID of the current document or folder to be saved.

itemidWorkflowId

This parameter is always NULL.

flsUnindexed

This value is TRUE if the object is a new document that has not been indexed in the system.

hOrigClass

Is the original class ID of the opened document or folder.

hClass

Is the current class ID of the opened document or folder. This value is the same as the *hOrigClass* parameter unless the user specifies a new index class.

szUserId[LST_USERID_LEN+1]

Is the user ID of the user saving the document or folder.

szUserHandle[LST_USERID_LEN+1]

This parameter is reserved.

usAccessLevel

Is the access privilege the user has for this document or folder. The valid value for this user exit routine is:

UX_PRIV_WRITE when the user opens this object in UPDATE mode.

sFields

Is the number of fields passed to the exit in the *pFields* parameter.

pFields

Is the pointer to an array of FIELDVALUE data structures. The configuration and content of the user-defined attributes for this document or folder are passed to the exit in these data structures.

```

FIELDVALUE:
typedef struct
{
    USHORT
        usFieldId;
    USHORT
        usDataType;
    USHORT
        usMaxLength;
    BOOL flsReq;
    PSZ pBuffer;
} FIELDVALUE;

```

```
typedef FIELDVALUE * PFIELDVALUE
```

where:

usFieldId

Is the user-defined attribute ID.

usDataType

Is IBM Content Manager for iSeries data type of the attribute in the *usFieldId* parameter. This is a numeric equivalent representing the data type. Refer to the section "Attribute types" in the frnpfi.h header file for the define statements and content requirements for these numbers.

usMaxLength

Is the maximum number of bytes in the *pBuffer* parameter to appear in the Index Form window, excluding the NULL terminator.

flsReq

This value is TRUE if the field is required. If this parameter is set to TRUE and this FIELDVALUE data structure is modified by the exit, the value in *pBuffer* must not be changed to NULL.

pBuffer

Is the current value of the attribute in ASCIIZ display format. The buffer length is the value in the *usMaxLength* parameter plus one to represent the NULL terminator.

Results

The function returns a value of SHORT with zero for SUCCESS. If any other value is returned, the Save operation is ended. An error message appears.

If the exit routine completes successfully, the error string pointer address in the *ppszErrorMsgs* parameter is interrogated. If the error string pointer is NULL or it points to a NULL error string, the attribute values returned in the FIELDVALUE data structures are audited by IBM Content Manager for iSeries against the data type, minimum, and maximum length requirements. Audit errors appear in the Index Errors window. A question mark appears beside each attribute field in the Index Form window with audit errors. In this case, the user must correct the errors and select the Save option again to save the record in IBM Content Manager for iSeries database.

The error string pointer in the *ppszErrorMsgs* parameter refers to the messages to appear. The format of the error message string is :

```
string1 (zero-terminated) <one or more zero-terminated
strings >zero terminator
```

These error messages appear in the Index Errors window. Each zero-terminated string appears on a new line in the window. Any audit errors found by IBM Content Manager for iSeries appear in the same Index Errors window.

If an error message string is returned by the exit, the pointer addressed in the *ppusFieldIdsInError* parameter must be set to point to an array of attribute IDs in error. There must be one attribute ID in this array for each message in the error string referred to by the *ppszErrorMsgs* parameter. The user-defined attribute name from the Index Form window appears to the left of its corresponding error message in the Index Errors window. A question mark appears next to the field on the Index Form window.

Comments

The exit routine must not free the buffers that are passed in. All items sent to the exit are read-only copies except the user-defined attributes in the FIELDVALUE data structures.

This exit is called when a document or folder is saved after modifying the user-defined attributes of the object. This exit is called prior to the Change System-Managed Storage user exit routine if both are specified for the current index class. Validation of the user-defined attribute fields is performed after the user exit routine is completed. IBM Content Manager for iSeries frees the error message buffer allocated by the exit after displaying the error messages to the user.

This exit is not processed if the Index Form window is not opened or if the user has not changed the class or attributes of the object.

UserActionUserExit (Workflow User Action User Exit)

Format

```
SHORT EXPENTRY UserActionUserExit( hSession, hWnd,  
pWorkManagementInfo, pExitStruct, pszAction )
```

The client application program calls this user exit when a user-defined action (function code 0050) is selected at a workbasket.

Use the Workflow Builder feature to specify this user exit and associate it with a user action function in an action list definition.

Parameters

hSession

HSESSION — input

Session handle returned by **SimLibLogon**.

hWnd HWND — input

Anchor window for message boxes. You can use this parameter to display messages and associate them with the application window.

pWorkManagementInfo

PWMSNAPSHOTSTRUCT — input

The pointer to a buffer where a WMSNAPSHOTSTRUCT data structure provides detailed work management information about the item selected.

pExitStruct

PUSEREXITSTRUCT — input

Contains index class and attribute information associated with the selected item.

pszAction

PSZ — input

The null-terminated character string containing the action selected. This is the value of the *ACTION variable.

Results

This user exit returns a value of type SHORT. It should return a value of zero for successful completion of the user exit. If it returns another value, an error message is displayed.

UserOptionUserExit (User-option User Exit)

Format

```
SHORT EXPENTRY UserOptionUserExit( hSession, hWnd, pExitStruct )
```

The client application program calls this user exit when a user-defined option is selected from the Selected menu for an item.

Use the system administration function to specify this user in the index class profile. Refer to the *IBM Content Manager for iSeries: System Administration Guide* for more information.

Parameters

hSession

HSESSION — input

Session handle returned by **SimLibLogon**.

hWnd HWND — input

Anchor window for message boxes. You can use this parameter to display messages and associate them with the application window.

pExitStruct

PUSEREXITSTRUCT — input

Contains index class and attribute information associated with the selected item.

This user exit returns a value of type SHORT. It should return a value of zero for successful completion of the user exit. If it returns another value, an error message is displayed.

WBItemSelectedUserExit (Workbasket Item Selected User Exit)

Format

```
SHORT EXPENTRY WBItemSelectedUserExit( hSession, hWnd, pWorkManagementInfo, pExitStruct, pfContinue )
```

The client application program calls this user exit when an item is selected at a workbasket. The exit is called before the item is displayed to the user.

Use the system administration function to specify this user exit in the workbasket profile. Refer to the *IBM Content Manager for iSeries: System Administration Guide* for more information.

Parameters

hSession

HSESSION — input

Session handle returned by **SimLibLogon**.

hWnd HWND — input

Anchor window for message boxes. You can use this parameter to display messages and associate them with the application window.

pWorkManagementInfo

PWMSNAPSHOTSTRUCT — input

The pointer to a buffer where a WMSNAPSHOTSTRUCT data structure provides detailed work management information about the item selected.

pExitStruct

PUSEREXITSTRUCT — input

Contains index class and attribute information associate with the selected item.

pfContinue

PBOOL — output

Pointer to the continue flag. Set this value to TRUE to continue with the display of the selected item. Set this to FALSE to bypass the display of the item.

Results

This user exit returns a value of type SHORT. It should return a value of zero for successful completion of the user exit. If it returns another value, an error message is displayed.

WBItemCompletedUserExit (Workbasket Item Completed User Exit)

Format

```
SHORT EXPENTRY WBItemCompletedUserExit( hSession, hWnd,  
pWorkManagementInfo, pExitStruct, pszAction, pfContinue )
```

The client application program calls this user exit when an action is selected at a workbasket that will complete working the item. The exit is called before the action is processed by the client.

Use the system administration function to specify this user exit in the workbasket profile. Refer to the *IBM Content Manager for iSeries: System Administration Guide* for more information.

Parameters

hSession

HSESSION — input

Session handle returned by **SimLibLogon**.

hWnd HWND — input

Anchor window for message boxes. You can use this parameter to display messages and associate them with the application window.

pWorkManagementInfo

PWMSNAPSHOTSTRUCT — input

The pointer to a buffer where a WMSNAPSHOTSTRUCT data structure provides detailed work management information about the item selected.

pExitStruct

PUSEREXITSTRUCT — input

Contains index class and attribute information associate with the selected item.

pszAction

PSZ — input

The null-terminated character string containing the action selected. This is the value of the SIMWM_ACTION variable.

pfContinue

PBOOL — output

Pointer to the continue flag. Set this value to TRUE to continue with the display of the selected item. Set this to FALSE to bypass the display of the item.

Results

This user exit returns a value of type SHORT. It should return a value of zero for successful completion of the user exit. If it returns another value, an error message is displayed.

UserDefinedWBUserExit (User-defined Workbasket User Exit)

Format

```
SHORT EXPENTRY UserDefinedWBUserExit( hSession, hWnd,  
pWorkBasketInfo, pszUserid )
```

The client application program calls this user exit when a user selects to open a workbasket of type 50 through 99. The user-defined workbasket type and this exit let you take advantage of the process control and workbasket control functions provided by Content Manager for iSeries. However, the interface to the workbasket and its contents are controlled by your own definition through this exit.

Use the system administration function to specify this user exit in the workbasket profile. Refer to the *IBM Content Manager for iSeries: System Administration Guide* for more information.

Parameters

hSession

HSESSION — input

Session handle returned by **SimLibLogon**.

hWnd HWND — input

Anchor window for message boxes. You can use this parameter to display messages and associate them with the application window.

pWorkBasketInfo

WORKBASKETINFOSTRUCT — input

The pointer to a buffer where a WORKBASKETINFOSTRUCT data structure provides detailed information about the user-defined workbasket.

pszUserID

PSZ — input

The null-terminated character string containing the user ID of the user calling this user exit.

Results

This user exit returns a value of type SHORT. It should return a value of zero for successful completion of the user exit. If it returns another value, an error message is displayed.

Server User Exits

The user exit points described here are invoked on the Content Manager for iSeries server.

Note: When calling the Content Manager for iSeries APIs from within any of the server exit points, the call must ensure that the SimLibLogoff API is called after the last API is called. Failure to do so may lead to unexpected results upon subsequent calls.

Content Manager for iSeries uses the OS/400 Registration Facility function to determine the exit programs to call. To add an exit program, enter the Work with Registration Information (WRKREGINF) command. On the Work with Registration Information screen, find the exit point and format name that you want to work with (see Table 1 for a list of the exit points and format names). Select option 8 (Work with Exit Programs) to work with exit programs for the specific exit point and format name. On the Work with Exit Programs screen do the following:

- If there is no program currently defined for the exit point, use option 1 (Add) to add an exit program entry. Enter a program number of 1 and the program name and library name for the program.
- If there is currently a program defined and you want to change the name of the program or the library, you must first remove the current entry using option 4 (Remove), then you must add the new program entry using option 1 (Add). Although the registration facility supports multiple exit programs, Content Manager for iSeries only supports one exit program per exit point.

If the Content Manager for iSeries exit points do not appear in the list, perform the following action from a command prompt to have them added:

```
CALL EKDCSUEREG PARM(' ' ' ')
```

After this call has completed, the list of exit points will be registered.

Table 4. Content Manager for iSeries Exit Points.

EXIT POINT NAME	FORMAT NAME	EXIT PROGRAM NAME
QIBM_VI-LOGON	VIF0100	User-defined
QIBM_VI_LOGOFF	VIF0100	User-defined
QIBM_VI_SAVE_ATTR	VIF0100	User-defined
QIBM_VI_CRT_OBJECT	VIF0100	User-defined
QIBM_VI_DLT_OBJECT	VIF0100	User-defined
QIBM_VI_OPEN_OBJECT	VIF0100	User-defined
QIBM_VI_CRT_ITEM	VIF0100	User-defined
QIBM_VI_ITEM_CREATED	VIF0100	User-defined
QIBM_VI_DLT_ITEM	VIF0100	User-defined
QIBM_VI_IMP_CREATED	VIF0100	User-defined
QIBM_VI_IMP_ITEM	VIF0100	User-defined
QIBM_VI_ADD_FLR_ITEM	VIF0100	User-defined
QIBM_VI_ROUTE_WP	VIF0100	User-defined
QIBM_VI_GET_WP	VIF0100	User-defined
QIBM_VI_RETURN_WP	VIF0100	User-defined
QIBM_VI_END_PROCESS	VIF0100	User-defined
QIBM_VI_SET_VARIABLE	VIF0100	User-defined

Logon User Exit

This user exit is called when a request is made to logon to Content Manager for iSeries using **SimLibLogon**.

Table 5. Logon User Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
User identifier	The user ID of the user to Log on.	Input	Character	10
Address	Workstation name or address.	Input	Character	15

Logoff User Exit

This user exit is called when a request is made to logoff of Content Manager for iSeries using **SimLibLogoff**.

Table 6. Logoff User Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
No parameters	This user exit is called when a request is made to logoff for Content Manager for iSeries using SimLibLogoff .			

Save Attributes User Exit

This user exit is called when a request is made to save changes to the attributes of a document or folder using **SimLibSaveAttr** or **SimLibCloseAttr**. This exit point is

before the attributes are actually updated. Given this, you may validate or modify attributes within the exit program. Modified attributes are not validated upon return from the exit.

This exit is invoked prior to privilege verification and input validation.

Table 7. Save Attributes Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Item ID	Identifier of the item for which attributes will be changed.	Input	Character	16
Index Class	An index class identifier for the set of user-defined attributes to associate with the item.	Input	Binary	4
Attributes	Table of attribute identifiers and values associated with the item.	Input/Output	Character	*
Return Value	Indicates how subsequent processing should continue. Valid values are: <ul style="list-style-type: none"> • 0 - Normal processing. The attributes will be modified. • non-zero - Error processing. The request to change the attributes should not be processed. 	Output	Binary	4

The format of the Attributes parameter is:

Table 8. Attributes

FIELD	TYPE
Number of attributes	Binary (4)
Attribute table	Char (*)

The attribute table consists of an array of attribute table entries. The number of entries in the attribute table is based on the value in the Number of Attributes parameter above.

Table 9. Attribute Table Entry

FIELD	TYPE
Attribute identifier	Binary (4)
Attribute type	Binary (4)
Attribute length	Binary (4)
Attribute value	Char (*)

Create Object User Exit

This user exit is called when a request is made to create an object using **SimLibCreateObject**. This exit point is after the create object request has been processed. Therefore, the item identifier and part number are the new object. This exit is invoked only if the create request was successfully processed.

Table 10. Create Object Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
-------	-------------	--------------	--------	------

Table 10. Create Object Exit Parameters (continued)

Item identifier	Item identifier of the object.	Input	Character	16
Part number	The part number of the object.	Input	Binary	4
Version	Version number of the object.	Input	Binary	4
Affiliated type	The type of affiliated values are: <ul style="list-style-type: none"> • SIM_ANNOTATIVE • SIM_BASE • SIM_EVENT • SIM_MGDS • SIM_NOTE 	Input	Binary	4

Delete Object User Exit

This user exit is called when a request is made to delete an object using **SimLibDeleteObject**. This exit point is after the delete request has been processed.

Table 11. Delete Object Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Item identifier	Item identifier of the object.	Input	Character	16
Part number	The part number of the object.	Input	Binary	4
Delete option	Valid delete options are: <ul style="list-style-type: none"> • SIM_DELETE_ITEM — Delete item if no more parts left. • SIM_DELETE_OBJECT — Don't delete the item, even if no more parts are left. 	Input	Binary	4
Return code	Return code after processing delete object requests.	Input	Binary	4

Open Object User Exit

This user exit is called when a request is made to open an object using **SimLibOpenObject**. This exit point is called prior to the request being processed.

Table 12. Open Object Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Item identifier	Item identifier of the object.	Input	Character	16
Part number	Part number of the object.	Input/Output	Binary	4
Access Level	Type of Access given to the object when opened. Valid values are: <ul style="list-style-type: none"> • SIM_ACCESS_READ_WRITE • SIM_ACCESS_SHARED_READ 	Input/Output	Binary	4

Table 12. Open Object Exit Parameters (continued)

Return Value	Indicates how processing will continue. 0 Normal processing. The object will be opened. Non-zero Error processing and the request to open will not be processed and this return value will be returned to the user.	Output	Binary	4
--------------	---	--------	--------	---

Create Item User Exit

This user exit is called when a request is made to create an item using **SimLibCreateItem**. This exit point is before the item is created. Given this, you may validate or modify the index class and associated attributes within the exit program.

This exit is invoked prior to privilege verification and input validation.

Table 13. Create Item Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Item type	The type of item you want to create. The valid values are: <ul style="list-style-type: none"> • SIM_DOCUMENT - Indicates that the item is a document. • SIM_FOLDER - Indicates that the item is a folder. 	Input	Binary	4
Index class	An index class identifier for the set of user-defined attributes to associate with this item.	Input/Output	Binary	4
Attributes	Table of attribute identifiers and values associated with the item created.	Input/Output	Character	*
Return value	Indicates how subsequent processing should continue. Valid values are: <ul style="list-style-type: none"> • 0 - Normal processing. The item will be created. • non-zero - Error processing. The request to create an item should not be processed. 	Output	Binary	4

See Table 8 on page 282 for a definition of the Attributes parameter.

Item Created User Exit

This user exit is called when a request is made to create an item using **SimLibCreateItem**. This exit point is after the item has been created.

Table 14. Item Created Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
-------	-------------	--------------	--------	------

Table 14. Item Created Exit Parameters (continued)

Item type	The type of item that was created. The valid values are: <ul style="list-style-type: none"> • SIM_DOCUMENT - Indicates that the item is a document. • SIM_FOLDER - Indicates that the item is a folder. 	Input	Binary	4
Index class	An index class identifier for the set of user-defined attributes associated with this item.	Input	Binary	4
Attributes	Table of attribute identifiers and values associated with the item created.	Input	Character	*
Item ID	The identifier of the item created.	Input	Character	16

See Table 8 on page 282 for a definition of the attribute parameter.

Delete Item User Exit

This user exit is called when a request is made to delete an item using `SimLibDeleteItem`. This exit point is after the item has been deleted.

Table 15. Delete Item Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Item ID	The identifier of an item that was deleted.	Input	Character	16
Return code	Return code after processing delete item request.	Input	Binary	4

Object Import Create Item User Exit

This user exit is called when the object import function creates an item. This exit point is before the item is created. Given this, you may validate or modify the index class and associated attributes within the exit program.

This exit is invoked prior to privilege verification and input validation.

Table 16. Object Import Create Item Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Item type	The type of item you want to create. The valid values are: <ul style="list-style-type: none"> • SIM_DOCUMENT - Indicates that the item is a document. • SIM_FOLDER - Indicates that the item is a folder. 	Input	Binary	4
Index class	An index class identifier for the set of user-defined attributes to associate with this item.	Input/Output	Binary	4
Attributes	Table of attribute identifiers and values associated with the item created.	Input/Output	Character	*

Table 16. Object Import Create Item Exit Parameters (continued)

Return value	Indicates how subsequent processing should continue. Valid values are: <ul style="list-style-type: none"> • 0 - Normal processing. The item will be created. • non-zero - Error processing. The request to create an item should not be processed. 	Output	Binary	4
--------------	--	--------	--------	---

See Table 8 on page 282 for a definition of the Attributes parameter.

Object Import Item Created User Exit

This user exit is called when the object import function creates an item. This exit point is after the item has been created.

Table 17. Object Import Item Created Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Item type	The type of item that was created. The valid values are: <ul style="list-style-type: none"> • SIM_DOCUMENT - Indicates that the item is a document. • SIM_FOLDER - Indicates that the item is a folder. 	Input	Binary	4
Index class	An index class identifier for the set of user-defined attributes associated with this item.	Input	Binary	4
Attributes	Table of attribute identifiers and values associated with the item created.	Input	Character	*
Item ID	The identifier of the item created.	Input	Character	16

See Table 8 on page 282 for a definition of the attribute parameter.

Add Folder Item User Exit

This user exit is called when a request is made to add an item to a folder using **SimLibAddFolderItem**. This exit point is before the item is added to the folder, giving you the option of changing the destination folder.

This exit is invoked prior to privilege verification and input validation.

Table 18. Add Folder Item Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Folder ID	The identifier of the folder to which the item will be added.	Input/Output	Character	16
Item ID	The identifier of the item to be added to the folder.	Input	Character	16

Table 18. Add Folder Item Exit Parameters (continued)

Return value	Indicates how subsequent processing should continue. Valid values are: <ul style="list-style-type: none"> • 0 - Normal processing. The item will be added to the folder. • non-zero - Error processing. The request to add the item to the folder should not be processed. 	Output	Binary	4
--------------	--	--------	--------	---

Route Work Package User Exit

This user exit is called when a request is made to route a work package using **SimWmRouteWorkPackage**. This exit point is before the work package is routed, giving you the option of changing the destination workbasket.

This exit is invoked prior to privilege verification and input validation.

Table 19. Route Work Package Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Workbasket	Workbasket identifier.	Input/Output	Character	11
Work Package ID	Identifier of the work package that represents the work being done.	Input	Binary	4
Instance ID	The identifier of the work package instance that distinguishes one parallel path from another within the process.	Input	Binary	4
Priority	Priority of the work. The priority affects the work sequencing as the work package moves through a process. A larger number is a higher priority.	Input/Output	Binary	4
Continue	<ul style="list-style-type: none"> • 0 - Continue with normal route processing. • non-zero - All required processing was performed within the exit, bypass any additional processing. 	Output	Binary	4
Return value	If continue is non-zero, this is the error code to be returned.	Output	Binary	4

Get Work Package User Exit

This user exit is called when a request is made to get a work package using **SimWmGetWorkPackage**. This exit work order and work package ID/instance may be overridden.

This exit is invoked prior to privilege verification and input validation.

Table 20. Get Work Package Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Workbasket	Workbasket identifier.	Input	Character	11

Table 20. Get Work Package Exit Parameters (continued)

Work Order	Order used for selecting an entry from the workbasket. Valid values are: <ul style="list-style-type: none"> • SIMWM_ORDER_FIFO - Make selection based on first in, first out, order to return first available work package. • SIMWM_ORDER_LIFO - Make selection based on last in, first out order to return first available work package. • SIMWM_ORDER_PRIORITY - Make selection based on the work package priority to return first available work package. • SIMWM_ORDER_SYSTEM_NEXT - The server determines the work order and returns the next available work package. • SIMWM_ORDER_FIFO_NEXT - Make selection for the next available work package based on first in, first out (FIFO) order. • SIMWM_ORDER_LIFO_NEXT - Make selection for the next available work package based on last in, first out (LIFO) order. • SIMWM_ORDER_PRIORITY_NEXT - Make selection for the next available work package based on the work package priority. • NULL - If work package ID is specified, select this work package. If work package ID is 0, the server determines the work order and returns the first available work package. 	Input/Output	Binary	4
Work package ID	Identifier to the work package that represents the work being done.	Input/Output	Binary	4
Instance ID	Identifier of the work package instance that distinguishes one parallel path from another within the process.	Input/Output	Binary	4

Return Work Package User Exit

This user exit is called when a request is made to return a work package using **SimWmReturnWorkPackage**. This exit point is before the return work package request has been processed. The priority may be overridden.

This exit is invoked prior to privilege verification and input validation.

Table 21. Return Work Package Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
-------	-------------	--------------	--------	------

Table 21. Return Work Package Exit Parameters (continued)

Work package ID	Identifier of the work package that represents the work being done.	Input	Binary	4
Instance ID	Identifier of the work package instance that distinguishes one parallel path from another within the process.	Input	Binary	4
Priority	Priority of the work to perform. The priority affects the work sequencing as the work package moves through a process. A larger number is a higher priority.	Input/Output	Binary	4

End Process User Exit

This user exit is called when a request is made to end a work package on a route, using **SimWmEndProcess**. This exit point is before the work package is ended.

This exit is invoked prior to privilege verification and input validation.

Table 22. End Process Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Work package ID	Identifier of the work package that represents the work being done.	Input	Binary	4
Instance ID	Identifier of the work package instance that distinguishes one parallel path from another within the process.	Input	Binary	4
Continue	<ul style="list-style-type: none"> 0 - Continue with normal end processing. non-zero - All required processing was performed within the exit, bypass any additional processing. 	Output	Binary	4
Return value	If continue is non-zero, this is the error code to be returned.	Output	Binary	4

Set Variable User Exit

This user exit is called during workflow processing, when a variable is being interrogated. The process will first determine if the variable is one of the following:

- Process
- Index class
- An existing variable
- Key field

If the variable is none of the above, the process will assume that the variable is an external variable and call this user exit to get the variable value.

Table 23. Set Variable Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
-------	-------------	--------------	--------	------

Table 23. Set Variable Exit Parameters (continued)

Work package ID	Identifier of the work package that represents the work being done.	Input	Binary	4
Instance ID	Identifier of the work package instance that distinguishes one parallel path from another within the process.	Input	Binary	4
Variable name	Name of the variable to process.	Input	Character	10
Variable value	Value of the variable	Input/Output	Character	40
Return value	Indicates how subsequent processing should continue. 0 Normal processing. Create variable. Non-zero Error processing. The request to create the variable should not be processed.	Output	Binary	4

Server User Exit for Process Definitions

This user exit is called as a step in a process when the process definition includes a user exit node.

Unlike the previous server user exits, the user exit node user exit is not entered in the OS/400 Registration Facility. When you define the user exit node as part of your process definition in Workflow Builder, you specify the program and library to be called.

Table 24. User Exit Node Exit Parameters

FIELD	DESCRIPTION	INPUT/OUTPUT	FORMAT	SIZE
Work package ID	Identifier of the work package that represents the work being done.	Input	Decimal	10
Instance ID	Identifier of the work package instance that distinguishes one parallel path from another within the process.	Input	Decimal	5
Return value	If the return value is 0, processing continues with the next command in the route definition. If the return value has any other value, error message EKD-1111 is logged to the error log file and the next command in the route definition is processed.	Output	Decimal	4
Reserved	Reserved for future use.	Input	Character	512

Appendix A. Guidelines for Search Expressions

Included in this appendix are some guidelines to follow when you are searching a Content Manager for iSeries client application.

Logical Operators for Searches

The following are the valid logical operators in order of precedence:

- NOT or ^** Negate the condition that follows.
- AND or &** Both the preceding condition and the condition that follows must be true.
- OR or |** Either the preceding condition or the condition that follows is true.

The following examples illustrate the precedence rules.

W, X, Y, and Z represent expressions in the following string:

W OR X AND NOT Y AND Z

Using the default precedence rules, this string is the same as the following:

W OR (X AND (NOT Y) AND Z)

You can use parentheses to alter precedence and change the meaning of the string. For example:

(W OR X) AND NOT (Y AND Z)

Note: You can enter the logical operators in uppercase, lowercase, or mixed case.

Search Expressions

Each search expression takes the following form: Attribute Operator Value Element Meaning

Attribute

A character string of the following form:

A n n n

Where the fields have the following meanings:

- A** An attribute. You can enter attributes in uppercase, lowercase, or mixed case.
- n n n** A decimal attribute ID. This value identifies either a user-defined attribute or a system-defined attribute as it exists in Content Manager for iSeries.

Operator

A relational operator. You can enter operators in uppercase, lowercase, or mixed case. The following are the valid operators.

- | Operator | Meaning |
|-----------------|----------|
| EQ or == | Equal to |

LEQ or <=	Less than or equal to
GEQ or >=	Greater than or equal to
LT or <	Less than
GT or >	Greater than
NEQ or <>	Not equal to
IN	In a list of values
NOTIN	Not in a list of values
LIKE	Like
NOTLIKE	Not like
BETWEEN	Between two values
NOTBETWEEN	Not between two values

Value

A string value, a numeric value, or the value NULL.

You must enclose string values within quotation marks. Use two quotation marks together to specify a zero-length string. Use two blanks within two quotation marks to specify a string of two blanks. Note that neither a zero-length string or a string of two blanks is equivalent to the value NULL.

You can place a plus or a minus sign before a numeric value. Optionally, you can specify a numeric value as a string.

Use the reserved word null to specify the value NULL. You can specify the value NULL for the EQ and NEQ operators only. The following are examples of valid values:

```
"XXXXX"
null
"123"
+123
123
```

Note: The values "123", +123, and 123 are equivalent.

Relational Operators for Searches

When you use the following relational operators, you must specify value strings in certain special formats:

- BETWEEN
- NOTBETWEEN
- LIKE
- NOTLIKE
- IN
- NOTIN

When you use either the BETWEEN operator or the NOTBETWEEN operator, you must specify all value strings within an expression in the same format. The following are examples of valid expressions:


```

A1 BETWEEN 100 200
A51 BETWEEN '1995-01-01' '2020-09-29'
A49 BETWEEN '1900-01-01-00.00.00.000000' '1920-02-02-00.00.00.000003'
A50 BETWEEN '13.00.00' '17.00.00'
A2 NOTBETWEEN "FIRST" "LAST"

```

When you use either the LIKE operator or the NOTLIKE operator, use the percent sign (%) or the underscore character (_) in SQL format to specify searches for partial strings.

Specify the percent sign to match any character. For example, the following expression searches for any value that begins with the character S:

```
A3 LIKE "S%"
```

Specify the underscore character to match any character in a certain position. For example, the following expression searches for any value that begins with the character string PA, contains any character in the third position, and contains the character K in the fourth and final position:

```
A8 LIKE "PA_K"
```

When you use either the IN operator or the NOTIN operator, you must enclose string values within apostrophes (') and enclose the entire set of values within parentheses. Additionally, you must place a comma (,) between any two values within an expression. The following are examples of valid expressions:

```

A4 IN "('Monday', 'Tuesday', 'Wednesday')"
A50 NOTIN "('15.30.03') "
A51 NOTIN "('1994-08-31') "
A49 NOTIN "('1920-02-02-00.00.00.000001') "
A5 NOTIN "(1,3,5,7,9)"

```

If you specify any attribute in an expression that does not belong to the index class you specify for that expression in this data structure, the search method fails. In such a case, the function fails regardless of any other correctly structured portion of the expression.

In the following example, the function fails if the index class you specify contains only attribute 10 and attribute 12:

```
(A12 == 3) OR (A38 < 5)
```

The expression in the preceding example causes the method to fail because the index class you specify does not contain attribute 38.

If you specify a null string ("") as the value of the index class, the method automatically searches only the index classes that contain the attributes you specify in the expression within the search string. If that expression consists of system attribute IDs only, the function searches all current index classes.

Process/Location Search

Process and location are the only system-defined attributes which may be specified within a search. The associated attribute identifiers are SIM_INDEX_ATTR_PROCESS and SIM_INDEX_ATTR_LOCATION, respectively. If you would like to specify process and location within a search, the first search expression must contain the process criteria. Location is optional, but if specified, the second search expression must contain location criteria, including location type. The value element of the search expressions should contain a valid process or

location identifier. The search expressions should not contain an operator. Valid location types are SIMWM_WORKBASKET and SIMWM_COLLECTION_POINT. For example:

```
A-20 "PAPPLICANT " A-21 3 "WORK05 "
```

Appendix B. Predefined Content Classes

Table 25 lists the predefined content classes for Content Manager for iSeries.

Table 25. Predefined Content Classes

Content Class	Description
SIM_CC_ADVWRITE	HP AdvanceWrite Plus format
SIM_CC_AIX_EXE	AIX [®] executable program
SIM_CC_AIXCMD	AIX command file
SIM_CC_AMIPRO	Ami Pro format
SIM_CC_AOCA	Audio Object Content Architecture (AOCA) data only
SIM_CC_ASCII	Flat ASCII text
SIM_CC_BCOCA	Tiled Bar Code Object Content Architecture (BCOCA) data only
SIM_CC_BKMGR_READ	BookManager [®] Read format
SIM_CC_BINARY	Unformatted binary data
SIM_CC_DESCRIBE	DeScribe text editor
SIM_CC_DIGITAL	Digital DX and WPS-Plus format
SIM_CC_DWRITE	DisplayWrite [®]
SIM_CC_EBCDIC	Flat EBCDIC text
SIM_CC_ENABLE	Enable format
SIM_CC_EXCEL	Microsoft Excel
SIM_CC_FAXGRP3	Fax image in group 3 format
SIM_CC_FRN_NOTE	Application note log
SIM_CC_FRN_HISTORY	Application history log
SIM_CC_FWORK	Framework format
SIM_CC_GOCA	Graphic Object Content Architecture (GOCA) data only
SIM_CC_IBMFFT	DCA - Final Form text
SIM_CC_IBMWA	IBM Writing Assistant
SIM_CC_INTER	Interleaf Publisher format
SIM_CC_IOCA_FS11	Image Object Content Architecture (IOCA) data only
SIM_CC_IOCA_IRM	IRM version of IOCA, non-standard
SIM_CC_IOCA_TILED	Tiled IOCA only
SIM_CC_LEGACY	Legacy format
SIM_CC_MacWrite	MacWrite format
SIM_CC_MASS	MASS 11 format
SIM_CC_MGDS	IBM machine-generated data stream (MGDS) format (for forms, for example)
SIM_CC_RICHTEXT	Microsoft Rich Text format
SIM_CC_MODCA_FORM	Mixed Object Document Content Architecture (MO:DCA) form overlay structure

Table 25. Predefined Content Classes (continued)

Content Class	Description
SIM_CC_MODCA_IS2	MO:DCA-P document
SIM_CC_MODCA_PAGE	MO:DCA page structure only
SIM_CC_MSCRIPT	Lotus® Manuscript format
SIM_CC_MULTIMATE	Multimate** and Multimate/Advantage** format
SIM_CC_MSTSOFT	Mastersoft internal format
SIM_CC_OFSWRITE	Office Writer
SIM_CC_OS2EXE	OS/2® Version 2 executable program
SIM_CC_OS2CMD	OS/2 Version 2 command file
SIM_CC_OS2DLL	OS/2 Version 2 Dynamic Link Library (DLL)
SIM_CC_OS2V12_BMP	OS/2 Version 1.2 bitmap
SIM_CC_OS2V13_BMP	OS/2 Version 1.3 bitmap
SIM_CC_OS2V2_BMP	OS/2 Version 2.0 bitmap
SIM_CC_PCX	PCX
SIM_CC_PEACH	PeachText 5000 format
SIM_CC_PFS	PFS:First Choice format
SIM_CC_POSTSCRIPT	PostScript data
SIM_CC_PPDS	Printer data stream
SIM_CC_PRS	Freelance presentation
SIM_CC_PWRITE	Professional Write format
SIM_CC_QAWRITE	QA Write format
SIM_CC_QUATTRO	Quattro Pro format
SIM_CC_RFILE	Rapid File format
SIM_CC_RFT	IBM RFT:DCA
SIM_CC_TARGA	TARGA
SIM_CC_TEXT	Text (where code page is unknown or variable)
SIM_CC_TIFF_G3_FINE	Tagged Image File Format (TIFF) header, higher resolution fax
SIM_CC_TIFF_G3_STANDARD	TIFF header, standard fax
SIM_CC_TIFF_IRM	IRM version of TIFF, single page
SIM_CC_TIFF_SINGLE_STRIP	Raster in a single strip
SIM_CC_TIFF5	TIFF V5, multi-page allowed
SIM_CC_TIFF5_PAGE	TIFF V5, single page
SIM_CC_TIFF6	TIFF V6, multi-page
SIM_CC_TIFF6_PAGE	TIFF V6, single page
SIM_CC_TOTALWORD	Total Word format
SIM_CC_UNIPLEX	Uniplex onGo format
SIM_CC_UNKNOWN	Content class unknown
SIM_CC_USER	Start of user-defined content classes
SIM_CC_VKS	Volkswriter format
SIM_CC_WANGPC	WANG PC format

Table 25. Predefined Content Classes (continued)

Content Class	Description
SIM_CC_WG1	Graphics, from Lotus 1-2-3/G
SIM_CC_WINV3_BMP	Microsoft Windows Version 3 bitmap
SIM_CC_WINWRITE	Windows Write format
SIM_CC_WKS	Lotus spreadsheet format
SIM_CC_WORD	Microsoft Word format
SIM_CC_WORDSTAR	Wordstar format
SIM_CC_WP	WordPerfect format
SIM_CC_WRITENOW	WriteNow format
SIM_CC_XYWRITE	XyWrite format

Appendix C. External References

Many Content Manager for iSeries customers have other repositories of data on their iSeries or within their network, and would like the ability to access that data through the Content Manager for iSeries Windows client and programming interfaces. These *external documents* should be treated exactly the same as a Content Manager for iSeries document, including search, addition to folders, and inclusion in workflow. The end user should not need to know the location of the document, or know that the content is not managed by Content Manager for iSeries.

To satisfy the requirement to access external documents as if they were Content Manager for iSeries documents, support for External References is being made available. An External Reference is simply the indexing information that Content Manager for iSeries already uses, plus the location information needed by another application to retrieve the document content. In the simplest form, this might be the path name of a file stored in the iSeries file system or on a network drive accessible to workstations.

To define an External Reference to Content Manager for iSeries, a file must first be created containing the location information and the Content Manager for iSeries indexing information such as the Index Class, Key Fields, and Content Class. Each record in this file represents one document that is to be indexed into Content Manager for iSeries. By indexing all documents in the file at once, instead of calling the Content Manager for iSeries APIs for each document, processing time is minimized. On a model 510, indexing 1000 documents takes approximately seven seconds.

Four types of External References are supported:

- An OS/400 file
- A workstation (or network) file
- Data retrieved by a program called on the server
- Data retrieved by a program called on the workstation

After indexing the External References, they can be accessed through the Content Manager for iSeries APIs. These APIs are used today by the Content Manager client, the Content Connect client, and applications written by business partners and customers. These applications will now be able to transparently access documents stored in other locations, but indexed by Content Manager for iSeries.

The Content Class capability of the Content Manager Client is key to this solution. When the Content Manager Client opens a document, the Content Class associated with the document controls whether it will be displayed by the Content Manager Viewer, or passed to another application. For example, if a video or audio clip is imported, the user would identify the Content Class as AVI. When the document is opened, the Content Manager Client would start MPLAY32 to play the video. This makes it possible for any type of document to be indexed by Content Manager for iSeries, located through search interfaces, and displayed either by the Content Manager Viewer or an alternate program.

There are many uses for External References. For example, it would be possible to store a large number of documents (images, video clips, text, and so forth) on a CD or DVD, duplicate it for all users, then index those documents into Content

Manager for iSeries. By storing the path name to each of the files, users could quickly retrieve a document even if they are using a dial-up connection. The same indexing approach could be used for files on a iSeries or on a LAN drive.

For even more flexibility, a program can be called on either the workstation or iSeries to retrieve the data. For example, the document that is indexed into Content Manager for iSeries might refer to an employee record. The called program could gather information from multiple databases and prepare a simple text representation, an image, or even an HTML document that is returned to the workstation. The Content Manager client, using the Content Class, would either display the results directly, or pass the document to another program for display.

With support for External References, any information can be indexed by Content Manager for iSeries, located, managed, and displayed through the Content Manager Client. You now have the option to maintain a single, central index of all your enterprise documents, and increased flexibility for constructing documents dynamically.

Creating External References

To index one or more documents as external references, create records in the file EKD0314, and then run the indexing program QVIXRFINX. The following fields are defined in EKD0314:

INDEX CLASS

This is the name (not description) of the Content Manager for iSeries index class into which the document is to be indexed. If the index class specified does not exist, it can be created later. (If the index class does not exist, the documents cannot be located through the Content Manager for iSeries APIs or any application using the APIs.)

KEY1-KEY8 DATA

These fields contain the attributes for indexing the document. They will be written to EKD0312 (the indexing file) exactly as they appear in EKD0314.

CREATE DATE, CREATE TIME, USER ID

These fields will be written exactly as they appear.

CONTENT CLASS

For any document that type that can be processed directly by the Content Manager Client, use 0. For others, review EKD0318 to find an appropriate Content Class. If a Content Class does not exist, use DFU or another utility to define a Content Class. At this time, there is not an administrative interface for defining Content Classes.

EXTERNAL REFERENCE TYPE

Four types are supported:

- 1 The External Reference field contains information that is used by another program (the Object Handler) on the iSeries to retrieve the data. In this example, a fully qualified iSeries library, file, and member name is passed to the program QVIXRFSMP.
- 2 The External Reference field contains a fully qualified iSeries path. This may be a library/file/member as above, or the name of an IFS file.
- 3 The External Reference field contains information that is used by another program (the Object Handler) on the workstation to retrieve the data. The specified program must be a DLL containing the function name vi400extref with the following prototype:


```

int _Export _System vi400extref (
    HSESSION hSession, /* CM for iSeries session handle */
    char * extref, /* External reference string,
                  converted to workstation
                  code page. */
    FILE * filehandle) /* Handle of file to receive
                       document content. Must not
                       be closed. */

```

For a type 3 reference, the Object Handler is the name of a workstation DLL containing the following function. A non-zero return code will be returned as an error in SimLibOpenObject.

- 4 The External Reference field contains a fully qualified path name that can be accessed from the workstation.

EXTERNAL REFERENCE

The location data, either a file name or information to be passed to the Object Handler.

OBJECT HANDLER LIBRARY

For a type 1 reference only, the name of the iSeries library containing the Object Handler program.

OBJECT HANDLER PROGRAM

For a type 1 reference, the name of the iSeries program to receive the External Reference. This will be a standalone program that receives as input the following structure:

RCAREA **CHAR(8)**

Non-blank return code will be written to EKD0080 to indicate an error

FILENAME **CHAR(256)**

The content to be returned through the Content Manager for iSeries APIs must be written to the temporary file specified.

EXTREF **CHAR(256)**

The external reference, or location information, used to locate the document content

ITEM ID

This field should initially be blank. This field will be set to the Item ID created by the indexing process. When non-blank, the record is assumed to already be indexed, so will be skipped by QVIXRFINX.

After creating EKD0314, the indexing program QVIXRFINX may be run. This program can be called from a command line or another program. All required files are opened, a sufficient number of document IDs reserved, and each document is indexed. If there is any program failure, QVIXRFINX may be restarted and only those records which have not already been indexed will be processed.

Limitations: Documents are indexed using this batch approach to provide the best possible performance. At this time, there is no API provided to index such documents from another application. There is no security checking, so only selected users should be given authorization to QVIXRFINX. Fields are not validated during processing.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM

Advanced Peer-to-Peer Networking

AIX

Application System/400

APPN

AS/400

BookManager

C/400

CICS

COBOL

DisplayWrite

iSeries

Operating System/2

Operating System/400

OS/2

OS/400

Redbooks

RPG/400

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Lotus and 1-2-3 are registered trademarks of Lotus Development Corporation in the United States, other countries or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations used in this book and the product document library. Refer to the *IBM Dictionary of Computing*, ZC20-1699-09, for terms or abbreviations that do not appear here.

The following cross-references are used in this glossary:

- **Contrast with.** This refers to a term that has an opposed or substantively different meaning.
- **See.** This refers the reader to multiple-word terms in which this term appears.
- **See also.** This refers the reader to terms that have a related, but not synonymous, meaning.
- **Synonym for.** This indicates that the term has the same meaning as a preferred term, which is defined in the glossary.

A

access list. A list consisting of one or more individual user IDs or user groups and the *privilege set* associated with each user ID or user group. You use access lists to control user access to items in Content Manager for iSeries. The items that can be associated with access lists are *index classes*, *workbaskets*, and *processes*.

action list. An approved list of the actions, defined by a supervisor, that a user can perform while working with items in a workbasket.

ad hoc route. A route that is not part of a defined process. An *ad hoc route* is started when a user assigns an item directly to a workbasket. The user manually routes the item from one workbasket to another by reassigning it.

administrator. The person responsible for system management, controls, and security, as well as case statistics. Synonymous with system administrator.

advanced peer-to-peer networking (APPN). Data communications support that routes data in a network between two or more APPC systems that are not directly attached.

advanced program-to-program communications (APPC). Data communications support that allows programs on an iSeries server to communicate with programs on other systems having compatible communications support. This communications support is the iSeries method of using the SNA LU session type 6.2 protocol.

annotation. An added descriptive comment or explanatory note.

APAR. Authorized Program Analysis Report.

API. Application programming interface.

application programmer. A programmer who designs programming systems and other applications for a user's system.

application program interface (API). The formally-defined programming language interface which is between an IBM system control program or a licensed program and the user of the program.

APPC. Advanced program-to-program communications.

APPN®. Advanced Peer-to-Peer Networking®.

archiving. The storage of backup files and any associated journals, usually for a given period of time.

AS/400®. Application System/400®.

attribute. Used in Content Manager for iSeries APIs, a single value associated with an item (document or folder). Each index class can have up to eight attributes.

B

binary large object (BLOB). A large stream of binary data treated as a single object.

C

cartridge. (1) A storage device that consists of magnetic tape, on supply and takeup reels, in a protective housing. (2) For optical storage, a plastic case that contains and protects optical disks, permitting insertion into an optical drive. See also *optical disk* and *cartridge storage slots*.

cartridge storage slots. An area in an optical library where cartridges are stored.

collection. The definition of storage management controls associated with a group of objects that typically have similar performance, availability, backup, and retention characteristics.

collection point. (1) The point where work packages wait for specific events to either occur or become synchronized before processing can continue. (2) A collection point is part of a work process. For example,

a collection point is where work packages that are part of the “open a new account” work process must wait until credit information is verified. See also *decision point*.

content class. A number that indicates the data format of an object, such as MO:DCA, TIFF, or ASCII.

control files. Files that govern the categories of work performed by an operator and the types of documents the system recognizes.

convenience workstation. A display workstation equipped with a printer and a scanner.

current document. A document that is being processed.

customization. The process of designing a data processing installation or network to meet the requirements of particular users.

D

DASD. Direct access storage device.

DDM. Distributed data management.

DBCS. Double-byte character set.

decision point. (1) The point where work packages continue on their current route or switch to an alternate route, depending on the specific information in each work package. Decision points are tables consisting of variable names, values, and routes. (2) A decision point is part of a work process. For example, a decision point is where work packages that are part of the “open a new account” work process receive approval or not based on credit information.

See also *collection point*.

direct access storage device (DASD). A device in which access time is effectively independent of the location of the data.

distributed data management (DDM). A feature of the System Support Program that lets an application program work on files that reside in a remote system.

display workstation. An image processing workstation used primarily for displaying documents that have been previously scanned or imported into the iSeries server.

document. (1) An item containing one or more base parts. (2) A named, structural unit of text that can be stored, retrieved, and exchanged among systems and users as a separate unit. Also referred to as an *object*. A single document can contain many different types of base parts, including text, images, and objects such as spreadsheet files.

document content architecture (DCA). An architecture that guarantees information integrity for a document being interchanged in an office system network. DCA provides the rule for specifying form and meaning of a document. It defines revisable form text (changeable) and final form text (unchangeable).

double-byte character set (DBCS). A set of characters in which each character occupies two bytes. Languages, such as Japanese, Chinese, and Korean, that contain more symbols than can be represented by 256 code points, require double-byte character sets. Entering, displaying, and printing DBCS characters requires special hardware and software support.

E

export. A process used to write data from a document in a system folder to a file. Export and import processes can be used to transfer documents among systems.

F

first in first out (FIFO). A queueing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

folder. In Content Manager for iSeries, an object that can contain other folders or documents.

folder balancing. In the iSeries, the process by which documents are distributed evenly among the available folders in the system.

folder manager. In IBM Content Manager for iSeries systems other than Content Manager for iSeries, the term used to describe the data model and a subset of the APIs. In Content Manager for iSeries, this term refers to the entire set of Content Manager for iSeries APIs.

G

Group III. A compression algorithm that conforms to a standard promulgated by the International Telegraph and Telephone Consultative Committee (CCITT).

H

HTML. Hypertext markup language.

I

image. (1) A single page of information; the result of scanning, or digitizing, a single sheet of paper. (2) An electronic representation of a picture produced by means of sensing light, sound, electron radiation, or other emanations from the picture or reflected by the

picture. An image can also be generated directly by software without reference to an existing picture. See also *page image*.

image data. Rectangular arrays of raster information that define an image. Image data is often created originally by a scanning process.

image host. The system where scanned and imported documents are permanently stored. See also *optical library subsystem*.

Image Object Content Architecture (IOCA). A structured collection of constructs used to interchange and present images.

image workstation. A programmable workstation that can perform image functions.

importing. A process by which documents are input into iSeries using files rather than the scanning process. Imported documents can be stored in Content Manager for iSeries on DASD and optical, and displayed and printed, in the same manner as scanned documents.

inbound. Pertaining to communication flowing in a direction towards the application program from external sources, such as a transmission from a terminal to the application program. Contrast with *outbound*.

index. To associate a document or folder with an index class and provide the key field values required by that class.

index class. A category for storing and retrieving objects, consisting of a named set of attributes known as *key fields*. When you create an item in Content Manager for iSeries, your application must assign an index class and supply the key field values required by that class. An index class identifies the automatic processing requirements and storage requirements for an object.

instance. An occurrence of a work package within a process. If the process consists of parallel routes, multiple instances of a work package exist.

iSeries object directory profile. A control file used in Content Manager for iSeries to identify iSeries object directories used for image document storage.

item. (1) Set of attributes and objects—one or more files containing image data, annotations, notes, or other content—that together represent a physical document, such as an insurance claim or a folder.

See also *document*. (2) The smallest unit of information that the library server administers. An item can be a folder, document, workbasket, or process. Referred to as an *object* outside of library server functions.

K

key field. An attribute of an item that represents a type of information about that item. For example, a customer data item might have key fields for the customer's name and social security number.

keyword. A name or symbol that identifies a parameter.

L

LAN. Local area network.

language profile. A control file used in Content Manager for iSeries to define parameters that are specific to a territory, such as time and date formats.

last in, first out (LIFO). A queueing technique in which the next item to be retrieved is the item most recently placed in the queue.

library server. The component of Content Manager for iSeries that contains index information for the items stored on one or more *object servers*.

LIFO (last in, first out). A queueing technique in which the next item to be retrieved is the item most recently placed in the queue.

local area network (LAN). A computer network located on a user's premises within a limited geographical area.

LU 6.2. In Systems Network Architecture (SNA), a type of session between two application programs in a distributed processing environment, using the SNA character string or a structured-field data stream; for example, an application program using CICS® communication with an iSeries application.

M

Machine-Generated Data Structure (MGDS). Data extracted from an image and put into generalized data stream (GDS) format.

magnetic storage. A storage device that uses the magnetic properties of certain materials.

magnetic tape. A tape with a magnetizable layer on which data can be stored.

magnetic tape device. A device for reading or writing data from or to magnetic tape.

MGDS. Machine-Generated Data Structure.

Mixed Object: Document Content Architecture (MO:DCA). An IBM architecture developed to allow the interchange of object data among applications within the interchange environment and among environments.

Mixed Object: Document Content Architecture-Presentation (MO:DCA-P). A subset architecture of MO:DCA that is used as an envelope to contain documents that are sent to the Content Manager for iSeries workstation for displaying or printing.

MO:DCA. Mixed Object: Document Content Architecture.

MO:DCA-P. Mixed Object: Document Content Architecture-Presentation.

MRI. Machine-readable information.

N

national language support (NLS). The modification or conversion of a United States English product to conform to the requirements of another language or territory. This can include enabling or retrofitting of a product and the translation of nomenclature, MRI, or product documents.

network. An arrangement of programs and devices connected for sending and receiving information.

network table file. A text file created during installation that contains the system-specific configuration information for each node for each Content Manager for iSeries server. Each server must have a network table file that identifies it. The name of the network table is always FRNOLNT.TBL.

NLS. National language support.

O

object. (1) An item upon which actions are performed. A collection of data referred to by a single name.

The smallest unit within the system. For Content Manager for iSeries systems, this is typically a single-image document. (2) Any binary data entity stored on an object server. In the Content Manager for iSeries data model, *object* specifically refers to a document's contents or parts.

object authority. The right to use or control an object.

object directory. A control file used in Content Manager for iSeries to identify iSeries object directories used for image document storage.

object server. The component of IBM Content Manager for iSeries that physically stores the objects or information that client applications store and access.

operator. The person who handles daily system administrative tasks.

optical. Pertaining to optical storage.

optical cartridge. A storage device that consists of an optical disk in a protective housing. See also *cartridge*.

optical disk. A disk that contains digital data readable by optical techniques. Synonymous with digital optical disk.

optical drive. The mechanism used to seek, read, or write data on an optical disk. An optical drive may reside in an optical library or as a stand-alone unit.

optical libraries. Software used to store image data on optical platters. Only direct-attached optical systems contain optical libraries.

optical library subsystem. The hardware and software that provides the long-term storage of the image data. See also *image host*.

Optical Storage Support. Software that supports communication between stand-alone optical disk drives, the optical library, and Content Manager for iSeries. The software runs on the System/36™ 5363 unit serving as the optical controller.

optical system profile. A file used to define the optical controller used for the optical storage of documents.

optical systems. Hardware used to store image data on optical platters. Only direct-attach optical systems contain optical libraries.

optical volume. One side of a double-sided optical disk containing optically stored data.

OS/2. Operating System/2®.

OS/400. Operating System/400®.

outbound. Pertaining to a transmission from the application program to a device. Contrast with *inbound*.

override. A parameter or value that replaces a previous parameter or value.

P

page. A single physical medium; for example, an 8.5-inch by 11-inch piece of paper.

page image. The electronic representation of a single physical page. The bounds of a page image are determined by the electromechanical characteristics of

the scanning equipment, along with the image capture application specifications in the receiving data processing system.

page scan. The electromechanical process of scanning a physical page (paper) to create a bit image of the page.

pan. Progressively translating an entire display image to give the visual impression of lateral movement of the image.

PDF. Portable document format.

platter. See *optical disk*.

Presentation Text Object Content Architecture (PTOCA). An architecture developed to allow the interchange of presentation text data.

primary processor. In a group of processing units, the main processing unit and its internal storage through which all other units communicate.

printer workstation. A display workstation equipped with a printer.

priority. (1) A rank assigned to a task that determines its precedence in receiving system resources. (2) In Content Manager for iSeries workflow, the priority of the work to be performed. The priority affects the work sequencing of the work package. A larger number is a higher priority.

privilege. An authorization for a user to either access or perform certain tasks on objects stored in Content Manager for iSeries. The system administrator assigns privileges.

privilege set. In Content Manager for iSeries, collection of *privileges* for working with system components and functions. The system administrator assigns privilege sets to users (user IDs) and user groups.

process. The series of steps, events, and rules through which a work package flows. A process is a combination of the route, collection point, and decision point through which a predefined type or work package must progress.

For example, a process called "open new account" would describe the following:

- The steps that work packages related to opening a new account must follow
- The events (such as verifying credit information) that must occur before work packages for new accounts can be routed to another point in the system
- The decisions that determine whether to open a new account based on the information for that particular account (for example, a good credit rating versus a poor one).

process item. Item used as a building block in a work process.

profile. A file that governs the categories of work performed and the types of users recognized by the system.

program temporary fix (PTF). A temporary solution or bypass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.

PTF. Program temporary fix.

PTOCA. Presentation Text Object Content Architecture.

R

release. To remove suspend criteria from a work package so that it is available to be worked. A suspended work package is released when the criteria have been met, or when a user with proper authority overrides the criteria and manually releases pending requests.

render. To take data that is not typically image-oriented and depict or display it as an image. In Content Manager for iSeries, you can render word-processing documents as images for display purposes.

resolution. In computer graphics, a measure of the sharpness of the image, expressed as the number of lines and columns on the display screen or the number of pels per unit of area.

rotate. A function of the document display window and the scan document display window. The orientation depends on the option selected.

route. A set of steps that move work between workbaskets, collection points, and decision points.

S

SBCS. Single-byte character set.

scanner. A device that examines a spatial pattern one part after another and generates analog or digital signals corresponding to the pattern.

scanner workstation. A display workstation equipped with a scanner.

scanning. A physical process that enters documents into an Content Manager for iSeries workstation. After a document has been scanned, it can be stored permanently.

search criteria. In Content Manager for iSeries, the text string used to represent the logical search to be performed on the library server.

secondary processor. In a group of processing units, any processing unit other than the primary unit.

server. On a local area network, a data station that provides facilities to other data stations; for example, a file server, a print server, a mail server.

side by side. A function on the document display window that displays two pages of a multipage document next to each other.

single-byte character set (SBCS). A set of characters in which each character occupies one byte.

slot. (1) A position in a device used for removable storage media. (2) A space in an optical library where an optical cartridge is stored. See *optical cartridge*.

SMS. System-managed storage.

spool file. A file that holds output data waiting to be printed or input data waiting to be processed by a program.

staging. The process of moving a stored object from an off-line or low-priority device back to an on-line or higher priority device, usually on demand of the system or on request of a user. When a user requests an object stored in permanent storage, a working copy is written to the *staging area*.

stand-alone. Pertaining to an operation that is independent of any other device, program, or system.

storage. The action of placing data into a storage device.

storage class. A storage class, in combination with an optical system identifier, defines the set of optical volumes upon which documents can be stored. Documents with the same storage class and optical system ID are stored on the same optical volume.

storage method. A means of grouping documents together for storage to an optical disk.

storage system. A generic term for storage in Content Manager for iSeries.

subsystem. A secondary or subordinate system, or the programming support part of a system that is usually capable of operating independently of or asynchronously with a controlling system.

suspend. To hold a work package at a workbasket until stated criteria have been satisfied. Work packages can be suspended for multiple criteria, therefore multiple suspend requests can exist for a work package. A document work package can be suspended

for a specific date. A folder work package can be suspended for a specific date or index class.

system administrator. The person who manages the Optical Library Subsystem and the departmental processor. The system administrator helps with problem determination and resolution. Synonymous with *administrator*.

system-managed storage (SMS). The Content Manager for iSeries approach to storage management. The system determines object placement, and automatically manages object backup, movement, space, and security.

System Support Program (SSP). A group of IBM-licensed programs that manage the running of other programs and the operation of associated devices, such as the display station and printer. The SSP also contains utility programs that perform common tasks, such as copying information from diskette to disk.

T

tape. See *magnetic tape*.

tape cartridge. See *cartridge*.

U

user. Anyone requiring the services of Content Manager for iSeries. This term generally refers to users of client applications rather than the developers of applications, who use the Content Manager for iSeries APIs.

user exit. (1) A point in an IBM-supplied program at which a user exit routine may be given control. (2) A programming service provided by an IBM software product that may be requested during the processing of an application program for the service of transferring control back to the application program upon the later occurrence of a user-specified event.

user exit routine. A routine written by a user to take control at a user exit of a program supplied by IBM.

user ID profile. A file that contains one entry for each user. The entries contain information such as processing eligibility.

V

volume. A certain portion of data, together with its data carrier, that can be handled conveniently as a unit.

W

workbasket. A container that holds work packages. Workbaskets can be used as parts of process definitions

or ad-hoc routes. In Content Manager for iSeries, a logical location within the Content Manager for iSeries system to which work packages can be assigned to wait for further processing.

A workbasket definition includes the rules that govern the presentation, status, and security of its contents.

workflow. A system that lets an enterprise define a work process and environment to automate workflow and control business processes.

work order. The sequence of work packages in a workbasket.

work package. The work that is routed from one location to another. Users access and work with work packages through workbaskets.

work process. In work management, the series of steps, events, and rules through which a work package flows. A work process is a combination of the route, collection point, and decision point through which a work package must progress.

workstation. A computer processor unit, image display unit, scanners, and printers with which the user performs input, indexing, and printing.

Index

A

access
 ending 236
 getting 237
Access to Items
 Restricting 9
Access to the Client for Windows 204
add an item to a folder
 (SimLibAddFolderItem) 12
Add Folder Item User Exit 286
AFFTOCENTRYSTRUCT 133
Alternate Search User Exit 249
annotations 15, 29, 41
ANNOTATIONSTRUCT 134
APIs 11
application object 177
application programming interfaces 11
application programming interfaces
 (APIs) 1
Applications
 Compiling and Linking Content
 Manager for iSeries 11
Argument Types
 Property and 176
attribute information, getting
 (SimLibGetAttrInfo) 38
attribute set, closing
 (SimLibCloseAttr) 22
attributes
 listing attributes of an index
 class 227
 listing folder attributes 225
ATTRINFOSTRUCT 135
ATTRLISTSTRUCT 137

B

base object
 listing a content class 230

C

Case-Sensitivity 8
catalog an object
 (SimLibCatalogObject) 15
change SMS criteria for an object
 (SimLibChangeObjectSMS) 21
Change System-Managed Storage User
 Exit 251
change the index class of an item
 (SimLibChangeIndexClass) 19
Changing an Items Index Class 8
Class
 Changing an Items Index 8
CLASSATTRSTRUCT 138
CLASSINDEXATTRSTRUCT 139
CLASSINDEXSTRUCT 140
CLASSINFOSTRUCT 141
Client for Windows 204

Close a Table of Contents
 Ip2CloseTOC 117
Close a TOC (Ip2CloseTOC) 117
close an attribute set
 (SimLibCloseAttr) 22
close an object (SimLibCloseObject) 23
collection
 items 201
Collection
 Documents 174
 Items 175
Common Data Structures 133
Compiling and Linking Content Manager
 for iSeries Applications 11
Components
 Content Manager for iSeries 2
Concepts
 Content Manager for iSeries 5
content class
 listing a base object's content
 class 230
content classes
 listing 222
Content Manager for iSeries Applications
 Compiling and Linking 11
CONTENTCLASSINFO 142
copy an object (SimLibCopyObject) 25
Create a Work Package
 SimWmCreateWorkPackage 92
create an item (SimLibCreateItem) 26
create an object (SimLibCreateObject) 29
Create Item User Exit 284
Create Object User Exit 282

D

Data Structures
 Common 133
delete an item (SimLibDeleteItem) 34
delete an object (SimLibDeleteObject) 36
Delete Item User Exit 285
Delete Object User Exit 283
Determine Next Workbasket User
 Exit 254
Determine Workflow User Exit 258
document
 creating a copy of 210
document base object
 exporting 219
 importing 220
document image
 displaying 216
document note logs 206
document object 186
Document Object 174
document view windows
 closing 209
documents
 scanning 239
Documents and Folders 7
Documents Collection 174

documents object 189

E

End Process User Exit 289
ending access to VHLPI functions 218
error object 191
Error Object 174
Errors
 Handling 175
events 15, 29, 41

F

folder
 adding an item to 205
 creating a folder 212, 213
 listing contents 223, 225
 removing an item from 238
folder management concepts 5
free memory (SimLibFree) 37

G

get a TOC (SimLibGetTOC) 49
get a TOC for item affiliates
 (SimLibGetItemAffiliatedTOC) 41
get attribute information
 (SimLibGetAttrInfo) 38
get index class information
 (SimLibGetClassInfo) 40
get item information
 (SimLibGetItemInfo) 43
get session type
 (SimLibGetSessionType) 49
Get the Updates to a Table of
 Contents 119
get TOC updates 119
Get Work Package User Exit 287
getting information about documents and
 folders 7

H

Handling Errors 175
HOBJ 143

I

ICVIEWSTRUCT 143
image object 192
index class
 listing all attributes of 227
 listing an item's index class 231
Index Class
 Changing an Items 8
index classes
 listing 229

- Index Form and Save Record User Exit 272
- Interface
 - Using the OLE Automation 173
- Ip2CloseTOC 117
- Ip2CloseTOC (Close a Table of Contents) 117
- Ip2GetLibSessionInfo 118
- Ip2GetTOCUpdates 119
- Ip2ListAttrs 121
- Ip2ListContentClasses 122
- Ip2ListServers 123
- Ip2QueryClassPriv 124
- Ip2QueryPrivBuffer 125
- Ip2TOCCount 130
- Ip2TOCStatus 131
- item
 - adding to a folder 205, 213
 - changing an index class 207
 - deleting 215
 - displaying using a library object window 217
 - listing attribute information 231
 - listing index class 231
 - removing from a folder 238
- item affiliates, getting a TOC for (SimLibGetItemAffiliatedTOC) 41
- Item Created User Exit 284
- item information, getting (SimLibGetItemInfo) 43
- item object 194
- Item Object 175
- ITEMINFOSTRUCT 144
- ITEMNAMESTRUCT 146
- items
 - searching 239
 - searching for 241
- Items
 - Restricting Access to 9
- items collection 201
- Items Collection 175
- items in TOC 130

L

- library object window
 - displaying an item 217
- LIBSEARCHCRITERIASTRUCT 147
- list content classes (Ip2ListContentClasses) 122
- list user-defined attributes (Ip2ListAttrs) 121
- logging off 56
- logging on (SimLibLogon) 58
- logical data model 5
- logoff 236
- Logoff User Exit 281
- logon 237
 - user id 220
- Logon User Exit 281
- Logon/Logoff with the Client for Windows 204

M

- Management
 - Understanding Work 5
- memory, freeing (SimLibFree) 37
- Migrating Objects 9

N

- NAMESTRUCT 149
- naming folders 8
- note attributes 69, 84
- notes 15, 29, 34
- notes, TOC of 41
- Notices 303

O

- object
 - application 177
 - document 186
 - documents 189
 - error 191
 - image 192
 - item 194
- Object
 - Application 174
 - Document 174
 - Error 174
 - Item 175
- Object Import Create Item User Exit 285
- Object Import Item Created User Exit 286
- object information 68
- Objects
 - Client for Windows 173
 - Migrating 9
 - Releasing 175
- OBJINFOSTRUCT 149
- OLE Automation Interface
 - Using the 173
- OLE Objects for Windows
 - Properties and Methods of 177
- open item attributes 61
- Open Object User Exit 283
- Overload Trigger User Exit 264

P

- Parameters and Variables
 - Visual Basic 203
- Process Information Data Structure 163
- Program
 - Sample Visual Basic 176
- Programming Interface for Visual Basic
 - Sample High-Level 203
- Programming Tips 175
- Properties and Methods of OLE Objects for Windows 177
- Property and Argument Types 176

Q

- query a privilege buffer 125
- query an object (SimLibQueryObject) 68
- query privileges 124

- Query Sort User Exit 268

R

- RCSTRUCT 151
- read an attribute (SimLibReadAttr) 69
- read an object 70
- Releasing Objects 175
- remove an item from a folder 72
- resize an object 73
- Restricting Access to Items 9
- retrieving information about documents and folders 7
- Return Work Package User Exit 288
- Route Work Package User Exit 287

S

- Sample High-Level Programming
 - Interface for Visual Basic 203
- Sample Visual Basic Program 176
- save an attribute 75
- Save Attributes User Exit 281
- Save Record User Exit 272
- search 76
- search query results 268
- searching
 - items 239, 241
- seek an object 79
- Server User Exits 280
- Set Variable User Exit 289
- Sim400ConvertCodepage 115
- SimLibAddFolderItem 12
- SimLibCatalogObject 15
- SimLibChangeIndexClass 19
- SimLibChangeObjectSMS 21
- SimLibCloseAttr 22
- SimLibCloseObject 23
- SimLibCopyObject 25
- SimLibCreateItem 26
- SimLibCreateObject 29
- SimLibDeleteItem 34
- SimLibDeleteObject 36
- SimLibFree 37
- SimLibGetAttrInfo 38
- SimLibGetClassInfo 40
- SimLibGetItemAffiliatedTOC 41
- SimLibGetItemInfo 43
- SimLibGetItemSnapshot 44
- SimLibGetItemType 46
- SimLibGetItemXref 47
- SimLibGetSessionType 49
- SimLibGetTOC 49
- SimLibGetTOCData 53
- SimLibListClasses 55
- SimLibLogoff 56
- SimLibLogon 58
- SimLibOpenItemAttr 61
- SimLibOpenObject 63
- SimLibOpenObjectByUniqueName 66
- SimLibQueryObject 68
- SimLibReadAttr 69
- SimLibReadObject 70
- SimLibRemoveFolderItem 72
- SimLibResizeObject 73
- SimLibSaveAttr 75

- SimLibSearch 76
- SimLibSeekObject 79
- SimLibStageObject 80
- SimLibStoreNewObject 81
- SimLibWriteAttr 84
- SimLibWriteObject 85
- SimWmActivateWorkPackage 87
- SimWmBeginProcess 88
- SimWmChangeVariables 90
- SimWmCreateWorkPackage 92
- SimWmEndCollectionPoint 93
- SimWmEndProcess 94
- SimWmGetActionListInfo 95
- SimWmGetWorkBasketInfo 98
- SimWmGetWorkPackage 99
- SimWmGetWorkPackagePriority 101
- SimWmListHistory 102
- SimWmListProcesses 103
- SimWmListWorkBaskets 104
- SimWmMatchEvent 105
- SimWmQueryVariables 107
- SimWmQueryWorkPackage 108
- SimWmReturnWorkPackage 109
- SimWmRouteWorkPackage 111
- SimWmSetWorkPackagePriority 112
- SimWmSuspendWorkPackage 114
- SMS 154
- SMS, changing criteria
 - (SimLibChangeObjectSMS) 21
- SNAPSHOTSTRUCT 155
- stage an object 80
- store a new object 81
- store an object (SimLibCatalogObject) 15
- Supporting 8
- Supporting Case-Sensitivity 8

T

- Table of Contents
 - Get the Updates to a 119
- Tips
 - Programming 175
- TOCENTRYSTRUCT 157
- Types
 - Property and Argument 176

U

- Understanding Workflow 5
- Updates to a Table of Contents 119
- User Exits 249
 - Alternate Search User Exit 249
 - Change System-Managed Storage User Exit 251
 - Determine Next Workbasket 254
 - Determine Workflow 258
 - Overload Trigger 264
 - Query Sort 268
 - Save Record 272
- USERACCESSSTRUCT 158
- UserActionUserExit 276
- UserDefinedWBUserExit 279
- USERLOGONINFOSTRUCT 159
- UserOptionUserExit 277
- Using Logon/Logoff with the Client for Windows 204

V

- Variables
 - Visual Basic Parameters and 203
- VbVhlAddFolderItem() 205
- VbVhlAdminItemNoteLog() 206
- VbVhlChangeItemIndex() 207
- VbVhlCloseDocViews() 209
- VbVhlCopyDoc() 210
- VbVhlCreateFolder() 212
- VbVhlCreateFolderAddItem() 213
- VbVhlDeleteItem() 215
- VbVhlDisplayDocView() 216
- VbVhlDisplayVItem() 217
- VbVhlDropFuncs () 218
- VbVhlExportDocObj() 219
- VbVhlGetVIUserID() 220
- VbVhlImportDocObj() 220
- VbVhlListContClasses() 222
- VbVhlListFolderItems 223
- VbVhlListFolderItemsAttr() 225
- VbVhlListIndexClassAttr() 227
- VbVhlListIndexClasses() 229
- VbVhlListItemCC() 230
- VbVhlListItemInfo() 231
- VbVhlListWBItems() 233
- VbVhlListWorkBaskets() 234
- VbVhlLoadFuncs() 235
- VbVhlLogoff() 236
- VbVhlLogon() 237
- VbVhlRemoveFolderItem() 238
- VbVhlScanDoc() 239
- VbVhlSearchAdv() 239
- VbVhlSearchItem() 241
- VHLPI functions
 - access 235
 - ending access to 218
- Visual Basic
 - Sample High-Level Programming Interface for 203
 - Sample Program 176
- Visual Basic Parameters and Variables 203

W

- WBItemCompletedUserExit 278
- WBItemSelectedUserExit 277
- Windows
 - Access to the Client for 204
- Windows Objects
 - Client for 173
- WMACTIONLISTFUNCSTRUCT 160
- WMACTIONLISTINFOSTRUCT 161
- WMHISTLOGENTRYSTRUCT 162
- WMLOCATIONINFOSTRUCT 162
- WMPROCESSINFOSTRUCT 163
- WMSNAPSHOTSTRUCT 164
- WMSUSPENDSTRUCT 166
- WMVARSTRUCT 167
- wokbasket
 - listing all names for 234
- workbasket
 - listing contents 233
- WORKBASKETINFOSTRUCT 168
- Workflow 5

- Workflow Location Information Structure 162
- write an attribute (SimLibWriteAttr) 84
- write an object 85



Program Number: 5722-VI1

SC27-1139-01

