IBM ImagePlus
VisualInfo for AS/400

Application Programming Guide and Reference

Version 4 Release 1

SC34-4586-00

IBM ImagePlus
VisualInfo for AS/400

Application Programming Guide and Reference

Version 4 Release 1

SC34-4586-00

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page iii.

First Edition (September 1997)

This edition applies to Version 4 Release 1 of the IBM ImagePlus VisualInfo for AS/400 licensed program, Program Number 5733-A18, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch serving your area. Publications are not stocked at the address given below.

A form for readers’ comments appears at the back of this publication. If the form has been removed, address your comments to:

Information Development
IBM Corporation
Department CGMD, Building 062
PO Box 12195
Research Triangle Park, NC 27709-2195 USA

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

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 SWS General Legal Counsel, IBM Corporation, Department TL3 Building 062, PO Box 12195, Research Triangle Park, NC 27709-2195. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may change this publication, the product described herein, or both. These changes will be incorporated in new editions of the publication.

Programming Interface Information

This book provides general-use application programming interfaces (APIs) for IBM ImagePlus VisualInfo for AS/400 (VisualInfo for AS/400). It documents general-use programming interface and associated guidance information provided for VisualInfo for AS/400.

General-use programming interfaces let customers write programs that obtain the services of VisualInfo for AS/400.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries:

- Application System/400, AS/400
- Bar Code Object Content Architecture, BCOCA
- BookManager
- CICS
- DisplayWrite
- FlowMark

About This Book

This book describes how to create or integrate image, workflow, or other applications into an VisualInfo for AS/400 system by using the VisualInfo Client Interfaces for Windows. These application programming interfaces (APIs) support client application development for VisualInfo for AS/400 in the Microsoft Windows environment. 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 VisualInfo for AS/400
- Tips for identifying application requirements as you create a VisualInfo for AS/400 application
- Ways to use the APIs to write image, workflow, or other applications that use VisualInfo for AS/400 APIs
- The terminology used with VisualInfo for AS/400

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 VisualInfo for AS/400 system or application, you need to understand how VisualInfo for AS/400 works and how to create new applications for or integrate existing applications with VisualInfo for AS/400. 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 VisualInfo for AS/400 implementations, you can use this books as a reference.

To successfully program with VisualInfo for AS/400, you need the following:

- Experience with the VisualBasic programming language
- Experience developing applications for Windows NT or 95 using the C programming language
- An understanding of what your application does and how to apply imaging technology and the VisualInfo for AS/400 library system to it
- Knowledge of online debugging techniques

How This Book Is Organized

This book contains the following information.

- Chapter 1, “Introducing VisualInfo for AS/400” on page 2, introduces the software and hardware components of VisualInfo for AS/400, and the APIs available with VisualInfo for AS/400.
- Chapter 2, “Using the OLE Automation Interface” on page 6, shows you how to enable another Windows-based application to log on to VisualInfo for AS/400 and perform various tasks within the Client for Windows using APIs that are based on OLE 2.0 Automation.
- Chapter 3, “VisualInfo for AS/400 Concepts” on page 12, introduces you to VisualInfo for AS/400 concepts and capabilities.
- Chapter 4, “Sample High-Level Programming Interface for Visual Basic” on page 17, shows you how to enable another Windows-based application to log on to VisualInfo for AS/400 and perform various tasks within the Client for Windows using APIs that are based on OLE 2.0 Automation.
- Chapter 5, “VisualInfo for AS/400 Application Programming Interfaces” on page 24, describes the VisualInfo for AS/400 common application programming interfaces.
- Chapter 6, “Common Data Structures” on page 151, describes the common data structures and database tables you can use to manipulate and manage objects and classes of objects.
- Chapter 7, “Properties and Methods of OLE Objects for Windows” on page 183, describes the properties and methods associated with all client application objects.
- Appendix A, “VisualInfo for AS/400 Terminology” on page 204, defines terms that have specific meanings in VisualInfo for AS/400, arranged by component.
- Appendix B, “Guidelines for Search Expressions” on page 207, gives you some guidelines to follow when you are searching the Client for Windows.
- Appendix C, “Using FlowMark with VisualInfo for AS/400” on page 211, provides information about integrating IBM FlowMark with VisualInfo for AS/400.
- Appendix D, “Data Structures and Definitions” on page 214, describes abstract data types, structures, and definitions that the VisualInfo for AS/400 client high-level programming interface uses.
- Appendix E, “Return Codes” on page 217, provides information about errors that the VisualInfo for AS/400 client high-level programming interface returns.
- Appendix F, “Predefined Content Classes” on page 220 lists the predefined content classes for VisualInfo for AS/400.

How to Use This Book

Use Chapter 1, “Introducing VisualInfo for AS/400” on page 2 and Appendix A, “VisualInfo for AS/400 Terminology” on page 204 to familiarize yourself with VisualInfo for AS/400. Refer to Chapter 3, “VisualInfo for AS/400 Concepts” on page 12 for conceptual information about how to use the VisualInfo for AS/400 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
Italic	Field names in data structures Names of books as references Parameter names in API functions Terms defined for the first time in the book
ITALIC UPPERCASE	The maximum length of a field
Bold Mixed Case	API function names (example: **SimLibLogon**)
BOLD UPPERCASE	Field values Parameter values

Where to Find More Information

The following IBM documents contain information that you might find helpful when using VisualInfo for AS/400.

For a list of other related publications, see “Bibliography” on page 223. Request copies of IBM publications from your IBM representative or from the IBM branch office serving your area.

IBM ImagePlus VisualInfo for AS/400

When you order VisualInfo for AS/400, you receive the following printed publications as part of the VisualInfo for AS/400 license.

- *IBM ImagePlus VisualInfo: Client for Windows User’s Guide*, SC31-9052
- *IBM ImagePlus VisualInfo for AS/400: Licensing Information*, GC34-4589
- *IBM ImagePlus VisualInfo for AS/400: Planning and Installation Guide*, GC34-4585
- *IBM ImagePlus VisualInfo for AS/400: System Administration Guide*, GC34-4583

The remaining books are shipped in softcopy format only.

- *IBM ImagePlus VisualInfo: Messages and Codes*, SC31-9065
- *IBM ImagePlus VisualInfo for AS/400: Application Programming Guide and Reference*, SC34-4586

IBM ImagePlus Workfolder Application Facility for AS/400

When you order the Workfolder Application Facility feature of VisualInfo for AS/400, you also receive the following printed publications.

- *IBM ImagePlus Workfolder Application Facility for AS/400: Planning and Installation Guide*, GC34-4624

- *IBM ImagePlus Workfolder Application Facility for AS/400: System Administration Guide*, GC34-4625

The remaining books are shipped in softcopy format only.

- *IBM ImagePlus Workfolder Application Facility for AS/400: Designing a Work Process*, SC34-4588
- *IBM ImagePlus Workfolder Application Facility for AS/400: API*, SC34-4590
- *IBM ImagePlus Workfolder Application Facility for AS/400: User's Guide*, SC34-4584
- *IBM ImagePlus Workfolder Application Facility for AS/400: User's Guide for the Work Management Builder*, SC34-4587

About the Softcopy Document Library

In addition to the printed books you receive as part of the VisualInfo for AS/400 product, the entire document library is available in softcopy. Shipped on a separate document tape or CD-ROM, the library includes books for the Workfolder Application Facility feature. You can also purchase printed copies of any book in the library.

The library is available in the following formats:

- Portable document format (PDF)
- Hypertext markup language (HTML)

Before you can view or print PDF and HTML documents, follow the installation instructions in either the *IBM ImagePlus VisualInfo for AS/400: Planning and Installation Guide* or *IBM ImagePlus Workfolder Application Facility for AS/400: Planning and Installation Guide*.

PDF Files

Using PDF files, you can easily view documents online, as well as select one or more pages that you want to print.

To view PDF documents, you must have a Web browser and the Adobe Acrobat Reader. If you do not already have this tool, you can obtain it free by following the instructions for download from the Adobe home page:

```
http://www.adobe.com/prodindex/acrobat/readstep.html
```

A variety of national language readers are also available.

Using the Adobe Acrobat Reader, you can do the following:

- View documents that resemble printed pages.
- Easily navigate through PDF books as you would a hardcopy document.
- Use the text search capability by selecting **Find** from the **Tools** pull-down menu.
- Easily print a single page, a range of pages, or an entire document.
- Customize page size, graphics, fonts, and font sizes.

To view or print books from PDF files using the Adobe Acrobat Reader, follow these steps.

1. Start the Adobe Acrobat Reader.
2. Select **Open** from the **File** pull-down menu.
3. Select the location where you installed the document library.
4. Select the book you want to view.

HTML Files

Using HTML files, you can use a Web browser of your choice to view and print HTML books. These resemble printed books yet provide convenient HTML links to help you navigate through the information.

To view or print books from HTML files, follow these steps.

1. Start your Web browser.
2. Select the **Open** or **Open file** choice (usually from the **File** pull-down menu).
3. Select the location where you installed the document library.
4. Select the book you want to view.

BookManager Files

BookManager files for the VisualInfo for AS/400 document library will be available in the first half of 1998 on CD-ROM as part of the AS/400 softcopy collection kit. To view or print books using BookManager, refer to the books that accompany your softcopy collection kit.

Contents

Programming Guide

Chapter 1. Introducing VisualInfo for AS/400

This overview explains the ways to implement VisualInfo for AS/400 and VisualInfo for AS/400 components. This information is a framework for you to use to determine how to make the most of the VisualInfo for AS/400 APIs as you create your applications. It includes an overview of the following VisualInfo for AS/400 components:

Client Application Program
The client application you use can be one of the client application programs delivered with VisualInfo for AS/400 or an application that you develop.

Image Services

VisualInfo for AS/400 provides services for image capture, display, and printing. Collectively, these services are referred to as *image services*. VisualInfo for AS/400 incorporates many of these services within the VisualInfo client application program.

VisualInfo for AS/400 APIs
VisualInfo for AS/400 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 VisualInfo for AS/400.

With VisualInfo for AS/400 you can develop a customized document management solution that includes a host server and information-processing capabilities for multiple media types. Using VisualInfo for AS/400, 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.

VisualInfo for AS/400 provides a wide variety of application programming interfaces (APIs) that let you use your business applications with VisualInfo for AS/400 and available or selected future facsimile processing and document capture complementary offerings.

VisualInfo for AS/400 offers tailorable document processing for both large and small organizations. VisualInfo for AS/400 lets users capture, store, and retrieve documents online and provides document, folder, and work management capabilities. VisualInfo for AS/400 also provides extensive data integrity and security.

VisualInfo for AS/400 consists Windows NT or 95 clients connected to an AS/400 server. It provides enterprise-wide access to document processing, storage, and management. That way, VisualInfo for AS/400 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 VisualInfo for AS/400

VisualInfo for AS/400 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 VisualInfo for AS/400.

Client-Server Relationship

Figure 1 shows the client/server relationship in VisualInfo for AS/400. VisualInfo for AS/400 consists of a client connected to one or more host servers.

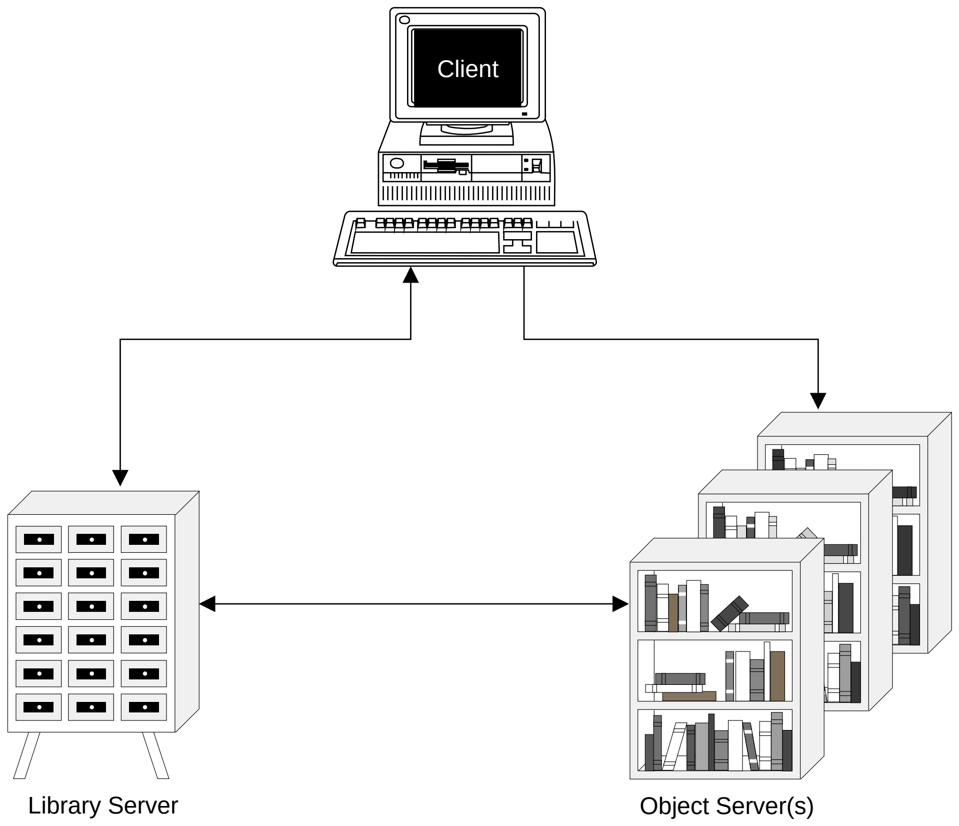

Figure 1. The Client-Server Relationship in VisualInfo for AS/400

The host server maintains document and folder index information, document and folder relationships, work-in-process information, and interacts with the client.

VisualInfo for AS/400 Components

VisualInfo for AS/400 consists of a client, which consists of the client application program, a host server, and VisualInfo for AS/400 APIs. You can use VisualInfo for AS/400 to develop additional clients.

The following figure shows the major components of VisualInfo for AS/400.

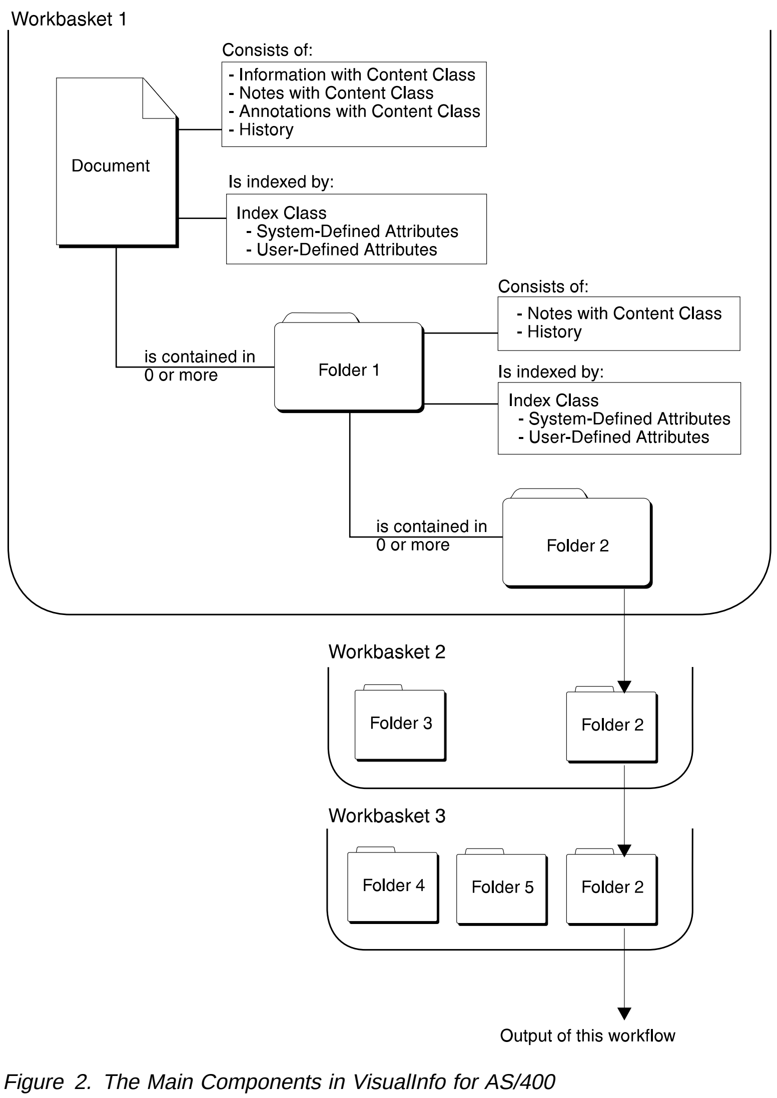
Workbasket 1
Consists of:
- Information with Content Class
- Notes with Content Class
- Annotations with Content Class
- History
Document
Is indexed by:
Index Class
- System-Defined Attributes
- User-Defined Attributes
Consists of:
- Notes with Content Class
- History
is contained in 0 or more
Folder 1
Is indexed by:
Index Class
- System-Defined Attributes
- User-Defined Attributes
is contained in 0 or more
Folder 2
Workbasket 2
Folder 3
Folder 2
Workbasket 3
Folder 4
Folder 5
Folder 2
Output of this workflow

Figure 2. The Main Components in VisualInfo for AS/400

Client Application

The VisualInfo for AS/400 client application provides document and folder management, scanning support, import and export, work management, and search capabilities built on the VisualInfo for AS/400 APIs.

The client application program provides a complete end-user interface for VisualInfo for AS/400. 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. After you configure the client application program, it becomes a complete document management application.

The client application program provides APIs to let you integrate folder, 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 VisualInfo for AS/400, write your own application, or use an application available from IBM Services or Business Partners.

VisualInfo for AS/400 APIs

If you choose to write your own application, you can use the VisualInfo for AS/400 APIs as the primary interface between the VisualInfo for AS/400 host server and your application.

In the VisualInfo for AS/400 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 *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 VisualInfo for AS/400. You can use these APIs to create a 32-bit Windows NT or 95 client application.

VisualInfo for AS/400 Server

The VisualInfo for AS/400 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 VisualInfo for AS/400 servers.

Chapter 2. Using the OLE Automation Interface

Using the API provided with the VisualInfo for AS/400 client, you can enable another Windows-based application to log on to VisualInfo for AS/400, 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 VisualInfo for AS/400 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 VisualInfo for AS/400 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 8 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 VisualInfo for AS/400, 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 VisualInfo for AS/400. You can open a TOC, which equates to a *Document* object in OLE automation, you can find or create items (Item objects), 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 Documents 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 VisualInfo for AS/400 return codes.

Image Object

The *Image* object represents a special document. It is the currently visible VisualInfo for AS/400 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 VisualInfo for AS/400 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 startup 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 the *Application Programming Reference, Volume 3* These types can be translated into VisualBasic 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_VARIENT (safe array)	Array (VB 4.0 or greater only)	IVarient*	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)	IVarient*	A structure used internally be 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 VisualInfo for AS/400
  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.

Chapter 3. VisualInfo for AS/400 Concepts

This section provides an overview of the VisualInfo for AS/400 concepts, including the logical data model. In other products of the IBM ImagePlus VisualInfo family, the term “folder manager” identifies a subset of application programming interfaces (APIs) and “common application programming interface” (CAPI) identifies a subset of *SimLib* interfaces. In VisualInfo for AS/400, all available programming interfaces are known as VisualInfo for AS/400 APIs.

Understanding the Logical Data Model

VisualInfo for AS/400 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 VisualInfo for AS/400 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. You can associate only one event part with a document. Annotation has a location, while a note does not.

Folders in VisualInfo for AS/400 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. You can place a folder in a workbasket as an item to be worked on and assign a folder to a workbasket.

Understanding Work Management

This release of VisualInfo for AS/400 implements a subset of the work management functions available in Workfolder Application Facility. In addition to VisualInfo for AS/400 concepts, work management introduces the following additional concepts:

Work package
Contains a single piece of work, such as a document or folder. A work package can be placed in one or more workbaskets.

Workbasket
Collection of work packages. The VisualInfo for AS/400 system administrator defines each workbasket so that it has a work order associated with it. When opening a workbasket, users are automatically get the next work package based on the order.

Work process
Set of rules for routing work packages through workbaskets. This release of VisualInfo for AS/400 supports only *ad-hoc routing*, in which the application running on the workstation routes a work package from one workbasket to the next.

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, workbasket 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:

```
ulRC = SimLibGetTOC(hSession,...);
if (ulRC != 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
       ulRC = SimLibGetTOCData(hSession,pTOC,i,NULL,pRC);
       if (ulRC != 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 view, your application already knows the index class of each item returned by the search. The index class contains the view you 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. (These two options are supported for 32-bit only.) 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

VisualInfo for AS/400 stores fixed- and variable-length character-string attributes exactly as presented by the application. VisualInfo for AS/400 does not convert to uppercase except for user ID and password.

Naming Folders

The folder data model for VisualInfo for AS/400 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 VisualInfo for AS/400 APIs, including **SimLibGetItemInfo** and **SimLibGetItemSnapshot** return this information to your application. You should use this index class within your application.

Restricting Access to Items

In the simplest example of access control, all users have access to all items in the library. To implement this type of access control, give all users maximum privileges and set all index class security ranges to '000'. However, there are many available levels of restricted access. One type of restriction is to keep a subset of users from seeing specific folders and documents.

The **SimLibLogon** function requires that a user have many privileges, including one to search for items, open items, and read attribute values. Users with the super access privilege can read any item in any index class. A typical user does not have the super access privilege unless the system is being run without access control (the configuration in which all users have access to all items). Therefore, the access list contents determine whether a typical user can see a protected item.

There are two primary methods for establishing restricted access to an item's attribute values and contents. The first is to assign security class ranges to index classes. The second method is to fine-tune the privilege settings. The system administration program lets you set individual library privilege bits to create a privilege set. You can turn on specific **LibItemSearch** privileges. Access control for both **SimLibOpenItemAttr** and **SimLibSearch** depend on the specific view. For **SimLibOpenItemAttr**, you must have access in the current view for the index class of the item. For **SimLibSearch**, you must have access in the view provided in the function parameters.

Nonadministrative users can use the privilege string *USER that comes with VisualInfo for AS/400. This privilege string includes all non-administrative selections from the system administration program.

SimLibLogon returns general privileges. **Ip2QueryClassPriv** returns privileges for index classes and index class views. 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.

When the object server receives a request to create an object, it returns the path name to the client. If the path is not accessible–such as when no assigned drive exists or the user does not have authority to the directory–the **SimLibCreateObject** call fails. Because files have already been updated on the object server, the VisualInfo for AS/400 API code sends a delete request to the server to remove the object entry and the item entry, if only one object exists. If the privilege string for the user does not allow deletion, the entries remain in the files. The result is an object that can never be opened. Therefore, you should consider giving delete authority to users who have create authority to prevent file entries that are not valid.

Migrating Objects

The VisualInfo for AS/400 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 4. Sample High-Level Programming Interface for Visual Basic

The VisualInfo for AS/400 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 VisualInfo for AS/400. Some highlights of the VisualInfo for AS/400 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 VisualInfo for AS/400 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 VisualInfo for AS/400 simultaneously.

General Use

The VisualInfo for AS/400 client high-level programming interface interacts with the basic components of the VisualInfo for AS/400 data model; documents, folders, and workbaskets. A VisualInfo for AS/400 document consists of a set of closely related objects or parts.

The VisualInfo for AS/400 client high-level programming interface provides functions to create, view, update and delete typical VisualInfo for AS/400 documents composed of a *single base part* (for example a scanned document or word processing file) and a *single note part*. Use of the VisualInfo for AS/400 high-level programming interface with documents containing multiple base parts can produce unexpected or undesired results. For additional information about the VisualInfo for AS/400 data model, see "Understanding the Logical Data Model" on page 12.

The Client for Windows's 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:

- **VhlApplObj** - Client for Windows's Application Object
- **VhlDocsObj** - Client for Windows's Documents Collection Object
- **VhlErrorObj** - Client for Windows's 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 onto, the following could be executed:

```
' Create Objects
ulRC = VbVhlLoadFuncs

' Get what server is logged on
Server$ = VhlApplObj.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 VisualInfo for AS/400. When started, this program logs on to VisualInfo for AS/400 and then wait for operator commands. Once logged on, other applications through the OLE automation interface can use the VisualInfo for AS/400 logon session established.

By using the Client for Windows's logon session, other applications do not need to logon to VisualInfo for AS/400, instead they must create an OLE automation Application Object from the Client for Windows. This can be done by executing the following:

```
Set VhlApplObj = CreateObject("Vic.Application")
```

where `VhlApplObj` 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 VisualInfo for AS/400.

Once the Client for Windows is logged onto VisualInfo for AS/400, 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 **VbVhlLogoff** 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 **VbVhlDropFuncs** should be used to terminate access only for the current application.

Programming Reference

Chapter 5. VisualInfo for AS/400 Application Programming Interfaces

This section describes the formats and parameters of the VisualInfo for AS/400 application programming interfaces (APIs). You can recognize these APIs by their **SimLib**, **SimWm**, and **Ip2** prefixes.

These APIs are strategic to the VisualInfo for AS/400 product. Although you can accomplish many functions through only the **SimLib** and **SimWm** APIs, most applications are likely to need at least some of the **Ip2** APIs.

All errors are logged in the VI400.LOG file.

The names of some VisualInfo for AS/400 data structures or symbolic constants might conflict with others that your applications use. If this is the case, please adjust your naming conventions to prevent conflicts.

For more information about the data structures for these APIs, see Chapter 6, “Common Data Structures” on page 151.

Compiling and Linking VisualInfo for AS/400 Applications

VisualInfo for AS/400 can be accessed through the VisualInfo for AS/400 APIs. You need the following files to build and run applications to access VisualInfo for AS/400:

EKDWSAPI.H
Work management API prototypes.

EKDWS.LIB
LIB file required to link with EKDWS.DLL.

EKDWS.DLL
All API functions.

EKDWS35I.DLL
IBM VisualAge runtime DLL.

EKDWSFRN.ERR
VisualInfo for AS/400 error numbers and descriptive names. The name is logged in VisualInfo for AS/400 for any error detected.

FRN*
VisualInfo for AS/400 header files.

These files are installed when you install the VisualInfo Windows Client Toolkit.

Applications must access headers as follows:

```
#include "EKDWSAPI.H"
```

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

The VisualInfo for AS/400 APIs use codepage conversion tables from VisualAge. Your installation program should install the required files for the codepages that are to be used for any given installation. The codepage 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 following 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**) unless another process on the same machine is using the file for logging. In that case, log records are intermingled but can be identified by the process number.

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 AS/400 system

VI400_LOG_STORAGE
VisualInfo for AS/400 object storage allocation and deallocation

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 VisualInfo for AS/400 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 example:

```
SERVER: MYVI400 REMOTE APPC
        LU_NAME     = USIBMNR.AS400DS1
        TP          = EKDCS01P.EKD310SRC
        MODE        = QPCSUPP
        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 AS/400 system. If the database name is not found in the network table, it is used as the symbolic destination name for CPI-C communications and must be defined in the side information section of the APPC communication product. Because the path in the VI400_CONFIG_PATH environment variable accesses FRNOLINT.TBL, it can be placed on a shared LAN drive or in a folder on an AS/400 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.

EKDWSFRN.ERR should be in the path defined in VI400_CONFIG_PATH. This file is used to log the descriptive name of each VisualInfo for AS/400 return code.

This version of the VisualInfo for AS/400 API set requires CPI-C level 1.2 support and Client Access/400 or an equivalent product that provides access through a shared folder. WCPIC32.DLL must be available in the path or logon will fail.

SimLibAddFolderItem (Add an Item to a Folder)

Format

SimLibAddFolderItem(*hSession, pszFolderID, pszItemID, pAsyncCtl, pRC* **)**

Purpose

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

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszFolderID
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.

pszItemID
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 *pszFolderID* parameter. You cannot add a folder to itself.

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 169.

Return Values

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

usParam
Not used

ulParam1
Not used

ulParam2
Not used

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_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>
#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"         /* VisualInfo for AS/400             */

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 *)pszItemID, (CHAR *) "F000000001");

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

   /*****************************************************************/
   /* Call SimLibCreateItem to create a new 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 VisualInfo for AS/400 functions:

- **SimLibCreateObject**
- **SimLibOpenObject**
- **SimLibWriteObject**
- **SimLibCloseObject**

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObj
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 162. “Guidelines for Use” describes the effects of your input to this data structure.

ulConCls
ULONG — input

The content class identifier for the object (see Appendix F, “Predefined Content Classes” on page 220). 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 VisualInfo for AS/400 content class services to determine how to manipulate the contents of objects that you store.

pSMS
PSMS — input

Pointer to a system-managed storage (SMS) structure for an object. This structure uses only *szCollectionName*.

pszFullFileName
PSZ — input

The pointer to a fully qualified directory path and file name

ulPriority
ULONG — input

Not supported.

fCreateControl
BITS — input

Control option bits for the cataloging operation. Here are the valid values:

SIM_CLOSE
Closes the object on completion of the request.

SIM_OPEN
Leaves the object open in update mode.

ulVersion
ULONG — input

Not supported.

lSeqAfterPart
LONG — input

Not supported.

ulAffiliatedType
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.

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 VisualInfo for AS/400 system ignores this parameter. For more information on the ANNOTATIONSTRUCT structure, see “ANNOTATIONSTRUCT (Annotation Information Structure)” on page 152.

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 169.

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 you specified SIM_OPEN as a flag in the *fCreateControl* 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.

ulRC
Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_INVALID_FOPTIONS
- 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.

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"                       /* VisualInfo for AS/400             */

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hItem
HITEM — input

The handle to a virtual item. The **SimLibOpenItemAttr** function returns this handle.

usClassId
USHORT — input

The identifier of the index class to change to.

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 169.

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_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 VisualInfo for AS/400 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.

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObj
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 162.

pSMS
PSMS — input

Pointer to a system-managed storage (SMS) structure for an object. This structure uses only *szCollectionName*.

fChangeControl
BITS — input

Not supported.

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 169.

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_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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hItem
HITEM — input

The handle to a virtual item. The **SimLibOpenItemAttr** function returns this handle.

ulDisposition
ULONG — input

The action to take regarding modifications to the item. The value of this parameter determines whether the VisualInfo for AS/400 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 VisualInfo for AS/400 system ignores this parameter. Here are the valid values:

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.

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 169.

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

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObjAcc
HOBJACC — input

The object access handle. The value of this parameter identifies the current instance of the accessed object.

fCommit
BOOL — input

The update commit flag. This flag applies only to objects open for update (SIM_ACCESS_READ_WRITE). Otherwise, the VisualInfo for AS/400 system ignores this flag. Here are the valid values:

TRUE or ***a nonzero value***
Closes and updates the timestamp of the object.

FALSE or 0
If no changes were made to the object, closes the object without changing the timestamp.

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 169.

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_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.

Example

```
   #include <stdio.h>                  /* Standard I/O header files         */
   #include "ekdviapi.h"                   /* VisualInfo for AS/400             */

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**

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 VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

usItemType
USHORT — input

The type of item you want to create. Here are the valid values:

SIM_DOCUMENT
Indicates that the item is a document.

SIM_FOLDER
Indicates that the item is a folder.

usIndexClass
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.

usNumOfAttrs
USHORT — input

The number of data structures in the *pAttributeList* parameter array.

pAttributeList
PATTRLISTSTRUCT — 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 155.

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.

ulAccessControl
ULONG — input

Not supported.

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 169.

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 an error occurs, this field contains the value 0.

ulParam1
Contains a PITEMID pointer to a buffer with the item identifier (*pszItemID*) for the new item.

ulParam2
The function does not use this field.

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"              /* VisualInfo for AS/400            */

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,                /* sessn 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.

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObj
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 162. “Guidelines for Use” describes the effects of your input to this data structure.

ulConCls
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 F, “Predefined Content Classes” on page 220). 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 VisualInfo for AS/400 content class services to determine how to manipulate the contents of the objects that you store.

pSMS
PSMS — input

Pointer to a system-managed storage (SMS) structure for an object. This structure uses only *szCollectionName*.

ulPriority
ULONG — input

Not supported.

fCreateControl
BITS — input

Control option bits for the creation operation. Here are the valid values:

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.

ulVersion
ULONG — input

Not supported.

lSeqAfterPart
LONG — input

Not supported.

ulAffiliatedType
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 VisualInfo for AS/400 system ignores this parameter. For more information on the ANNOTATIONSTRUCT structure, see “ANNOTATIONSTRUCT (Annotation Information Structure)” on page 152.

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 169.

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
- 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"                   /* VisualInfo for AS/400            */

 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**
- **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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemID
PITEMID — input

The identifier of an item you want to delete. This identifier is the item ID.

SimLibDeleteItem

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 169.

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 item is locked on the server, this field contains the value 1, to indicate that *ulParam1* contains a pointer.

ulParam1
If *usParam* 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. For more information on the USERACCESSSTRUCT data structure, see “USERACCESSSTRUCT (User Access Data Structure)” on page 178.

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_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 VisualInfo for AS/400 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 VisualInfo for AS/400 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"              /* VisualInfo for AS/400             */
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");

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

```
    /*****************************************************************/

     sResult = SimLibDeleteItem(
             hSession,                 /* sessn handle from SimLibLogon */
             pszItemID,                /* itemID to be deleted          */
             (PASYNCCTLSTRUCT) 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObj
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 162.

ulDeleteOption
ULONG — input

Not supported.

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 169.

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 locked, this field contains the value 1, to indicate that *ulParam1* contains a pointer. If the VisualInfo for AS/400 system returns any other error, this field contains the value NULL.

ulParam1
If *usParam* 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. For more information on the USERACCESSSTRUCT structure, see “USERACCESSSTRUCT (User Access Data Structure)” on page 178.

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_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 VisualInfo for AS/400 system. Do not call this function if your application allocated the memory. Use it only as directed.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pBuffer
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 169.

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 VisualInfo for AS/400 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 VisualInfo for AS/400 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 169.

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 153.

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**

SimLibGetClassIndexes (Get Class Indexes)

Format

SimLibGetClassIndexes(*hSession, usIndexClass, pAsyncCtl, pRC* **)**

Purpose

Use the **SimLibGetClassIndexes()** function to list each database index created for the index class you specify.

Parameters

hSession
HSESSION — input

The handle to the IBM ImagePlus VisualInfo for AS/400 session information. The **SimLibLogon()** function creates the session information.

usIndexClass
USHORT — input

The identifier of the index class for which you want a list of database indexes. Specify the index class 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 169.

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 array of CLASSINDEXSTRUCT data structures. For more information on this data structure, see “CLASSINDEXSTRUCT (Class Index Structure)” on page 159. If no index class indexes are found, this field contains the value NULL.

ulParam2
Contains the number of elements in the array pointed to by *ulParam1*, which is also the number of indexes for this index class. If no index class indexes are found, this field contains the value 0.

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_INDEX_CLASS
- SIM_RC_INVALID_MSGID
- SIM_RC_INVALID_PRC
- SIM_RC_IP2_ENV_VAR_NOT_FOUND
- SIM_RC_LEVEL2_DLL_LOAD_FAIL
- SIM_RC_NOT_SUPPORTED

Guidelines for Use

Exceptions

- Each index class has a default database index. However, this function does not return that default index and you cannot delete it using the **Ip2DeleteIndex()** function. The default database index is on the ITEMID column of the index class attributes table.
- An index class ID with the value 0 returns SIM_RC_INVALID_INDEX_CLASS.
- An index class that does not exist returns SIM_RC_OK, just as if it were a valid index class with no indexes.

Follow-Up Tasks

- Your application can use this information to prompt the user for those attributes that are indexed, leading to faster searches.
- When your application no longer needs the CLASSINDEXSTRUCT data, use the **SimLibFree(** *hSession,*(PVOID)*ulParam1, pRC* **)** function to free the buffer. One call to this function releases the CLASSINDEXSTRUCT data structure and any CLASSINDEXATTRSTRUCT data structures.

Related Functions

- **SimLibLogon()**

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 or index class view defined in the system.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

usClassType
USHORT— input

The type of information that the *usID* parameter contains. Here are the valid values:

SIM_INDEXCLASSID
Indicates that the *usID* parameter contains an index class ID.

usID
USHORT — input

The ID of an index class.

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 169.

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 the data area.

ulParam1
Contains a pointer to a buffer with a CLASSINFOSTRUCT data structure.

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_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

hSession
HSESSION — input

The handle to the IBM ImagePlus VisualInfo for AS/400 session information. The **SimLibLogon()** function creates the session information.

pszItemID
PITEMID — input

The identifier of an item for which you want a table of contents listing affiliated objects. This identifier is the item ID.

usAffiliatedType
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 169.

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 151.

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
- Work management information
- User ID of anyone who has locked the item

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemID
PITEMID — input

The identifier of an item for which you want information. This identifier is the item ID.

usClassId
USHORT — input

The identifier of an index class.

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 169.

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 an ITEMINFOSTRUCT data structure that provides the item information. For more information on this data structure, see "ITEMINFOSTRUCT (Item Information Structure)" on page 163.

ulParam2
Contains the value 1, indicating that the buffer referenced by *ulParam1* contains 1 entry.

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
- 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**
- **SimLibListClassess**

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 VisualInfo for AS/400 functions:

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

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemID
PITEMID — input

The identifier of an item. This identifier is the item ID.

fReadAttrInd
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 VisualInfo for AS/400 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 162 and “SNAPSHOTSTRUCT (Snapshot Information Structure)” on page 173.

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 169.

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
- 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 VisualInfo for AS/400 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemID
PITEMID — input

The identifier of an item for which you want to return the type. This identifier is 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 169.

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 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.

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_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 VisualInfo for AS/400 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemID
PITEMID — input

The identifier of an item for which you want a cross reference. This identifier is the item ID.

ulFilter
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.

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 169.

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 no items match the criteria you specify, this field contains the value NULL.

ulParam1
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.

ulParam2
Contains the number of entries pointed to by *ulParam1*.

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
- 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

hSession
HSESSION — input

The handle to the IBM ImagePlus VisualInfo for AS/400 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 169.

Return Values

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

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

ulParam1
Contains a PSZ to the current session type. If you have a LAN-based library session, the session type is lp2. Other values are platform dependent.

ulParam2
The function does not use this field.

ulRC
Contains one of the following return codes:

- 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemID
PITEMID — input

The identifier of workbasket or folder for which you want a table of contents. This identifier is the item ID.

usItemType
USHORT — input

The type of item to return in the table of contents. Here are the valid values:

SIM_DOCUMENT
Returns documents.

SIM_FOLDER
Returns folders.

SIM_ALL
Returns both documents and folders.

usWipFilter
USHORT — input

Not supported.

usSuspendFilter
USHORT — input

Not supported.

usNbrOfClasses
USHORT — input

The number of index class identifiers in the list you specify as the value of the *pusClassIdList* parameter. Specify the value 0 for the *usNbrOfClasses* parameter to indicate that class is not a criterion for selecting items.

pusClassIdList
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 *pusClassIdList* parameter only if you specify the value 0 for the *usNbrOfClasses* parameter.

pLinkCriteria
PVOID — input

Not supported.

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 169.

Return Values

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

usParam
Contains the number of items in the table of contents. If no items satisfy the filter, the field contains the value NULL.

ulParam1
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 176.

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.

ulParam2
Contains the table of contents handle (*hTOC*). If no items satisfy the filter, the 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_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"              // VisualInfo for AS/400
  HSESSION        hSession;        // Handle to a VisualInfo for AS/400 session
  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;

  ulRC = SimLibGetTOC(
            hSession,              // Handle to a VisualInfo for AS/400 session.
            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 (ulRC == 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 VisualInfo for AS/400 by using the  ... */
/* ... session handle obtained by calling SimLibLogon ... */
/*********************************************************/

ulRC = Ip2CloseTOC(
          hSession,             // Handle to a VisualInfo for AS/400 session
          hTOC,                 // TOC Handle from SimLibGetTOC
          NULL,                 //  by NULL, asynchronous call made
          &RC                   // pointer to return struct
          );
if (ulRC == 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pTOCEntries
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 176.

ulEntryCount
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.

fDataOptions
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.

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 169.

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 VisualInfo for AS/400 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 VisualInfo for AS/400 database. It lists only the classes for which this user has access and which contain attributes.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

fClassOptions
BITS — input

Not supported.

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 169.

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
If *ulParam2* 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 167.

ulParam2
Contains the number of fields in the array pointed to by *ulParam1*.

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_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 VisualInfo for AS/400 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 VisualInfo for AS/400 operations for a current application.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 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 169.

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_HSESSION
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC

Guidelines for Use

Effects

- After your application uses this function, any additional VisualInfo for AS/400 functions fail if they use the same session handle.
- All structures that a VisualInfo for AS/400 API allocates that are not released using **SimLibFree** are released during logoff.

Example

```
#include <stdio.h>                   /* Standard I/O header files    */
#include "ekdviapi.h"                    /* VisualInfo for AS/400        */

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.
            hSession       = (HSESSION)RC.ulParam1;
            pUserLogonInfo = (PUSERLOGONINFOSTRUCT)RC.ulParam2;
         } else {
            printf("error -SimLibLogon failed with %ld.\n",ulRC);
            exit(1);
         }

         /*****************************************************/
         /* Call other VisualInfo for AS/400 APIs by using the */
         /* session handle obtained by calling SimLibLogon     */
         /*****************************************************/

         /*****************************************************/
         /* Logoff from system, and end a normal session       */
         /*****************************************************/
         ulRC = SimLibLogoff(
                   hSession,           // Session handle
                   NULL,               // not supported
                   &RC            // pointer to return data struct
                   );
         if (ulRC == SIM_RC_OK) {
            /*****************/
            /* Logoff success */
            /*****************/
         } else {
            printf("error - SimLibLogoff failed with %ld\n.",ulRC);
            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 VisualInfo for AS/400 operations. Your application must use this function before it can use any other VisualInfo for AS/400 functions, and it must use the **SimLibLogoff** function when it has finished using VisualInfo for AS/400 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.

pszPassword
PSZ — input

The NULL-terminated character string that specifies the password for the user ID.

pszNewPassword
PSZ — input

Not supported.

pszProxyID
PSZ — input

Not supported.

pszProxyScope
PSZ — input

Not supported.

fSession
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.

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 169.

Return Values

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

usParam
Contains the value 0, to indicate that *ulParam1* contains a session handle and *ulParam2* contains a pointer to a buffer.

ulParam1
Contains an *hSession* parameter or NULL.

ulParam2
Not used.

ulRC
Contains one of the following return codes. “Guidelines for Use” contains more detail.

- SIM_RC_OK
- SIM_RC_COMMUNICATIONS_ERROR
- SIM_RC_INVALID_POINTER
- SIM_RC_INVALID_PRC
- 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"               /* VisualInfo for AS/400            */

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 VisualInfo for AS/400 APIs by using the */
/* session handle obtained by calling SimLibLogon     */
/*****************************************************/

/*****************************************************/
/* Logoff from system, and end a normal session      */
/*****************************************************/
ulRC = SimLibLogoff(
          hSession,                // Session handle
          NULL,                    // NULL indicates synchronous call
          &RC                      // pointer to return data struct
          );
if (ulRC == SIM_RC_OK) {
   /******************/
   /* Logoff success */
   /******************/
} else {
   printf("error - SimLibLogoff failed with %ld\n.",ulRC);
   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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemID
PITEMID — input

The identifier of an item that you want to open to access the attributes. This identifier is the item ID.

usClassId
USHORT — input

The identifier of an index class. This parameter is optional, but specifying the correct index class can improve performance. If the item has no index class or you do not know the index class, use the value 0.

ulAccessLevel
ULONG — input

The item access mode. The value of this parameter indicates the access mode for locking the item. Here are the valid values:

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.

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 169.

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 VisualInfo for AS/400 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. For more information on this data structure, see “USERACCESSSTRUCT (User Access Data Structure)” on page 178.

ulParam2
Returns the index class of the item. You can use this value to determine the index class if you specified the value 0 or an incorrect value in the *usClassId* parameter.

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

SimLibOpenItemAttr

Guidelines for Use

Effects

- If your application uses this function with read access, the VisualInfo for AS/400 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**.

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObj
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 162.

ulAccessLevel
ULONG — input

The object access mode. The value of this parameter indicates the access mode for opening the object.

Together, the *ulAccessLevel* parameter and the *fConflict* parameter establish an access state. The VisualInfo for AS/400 system uses this access state to accept or reject concurrent requests to access an open object. Here are the valid values:

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.

ulPriority
ULONG — input

Not supported.

fConflict
BOOL — input

Not supported.

fOpenControl
BITS — input

Not supported.

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 169.

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 *hObjAcc*, an HOBJACC object access handle. The value in this field identifies the current instance of the accessed object.

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_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*), 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"                    /* VisualInfo for AS/400             */

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

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObj
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 162.

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 169.

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 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 167.

ulParam2
The function does not use this field.

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_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 data in an OBJINFOSTRUCT data structure to provide the following information about the object that you specify.

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hItem
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.

usAttributeId
USHORT — input

The unique identifier assigned to an attribute.

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 169.

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 an error occurs, this field contains the value 0.

ulParam1
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.

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_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* **)**

SimLibReadObject

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 VisualInfo for AS/400 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.

ulBytesToRead
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.

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 169.

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 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.

ulParam2
Contains the actual number of bytes read.

ulRC
Contains one of the following return codes:

- 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszFolderID
PITEMID — input

The identifier of a folder from which you want to remove an item. This identifier is the item ID of the folder.

pszItemID
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.

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 169.

Return Values

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

usParam
If the return code is SIM_RC_PARENT_CHECKEDOUT, this field contains the value 1 to indicate that *ulParam1* contains a pointer.

ulParam1
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. For more information on the USERACCESSSTRUCT structure, see “USERACCESSSTRUCT (User Access Data Structure)” on page 178.

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_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.

- When deleting a folder, the function removes all items from the folder. The items themselves–documents or folders–are not deleted.

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* **)**

SimLibResizeObject

Purpose

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

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObjAcc
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.

ulSize
ULONG — input

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 169.

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hItem
HITEM — input

The handle to a virtual item. The **SimLibOpenItemAttr** function returns this handle.

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 169.

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_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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszItemFilter
PITEMID — input

Not supported.

pLinkCriteria
PVOID — input

Not supported.

usStatDyn
USHORT — input

Not supported.

usTypeFilter
USHORT — input

The type of items to search for. Here are the valid values:

SIM_DOCUMENT
Searches for documents.

SIM_FOLDER
Searches for folders.

SIM_FOLDER_DOC
Searches for both folders and documents.

fWipFilter
BITS — input

Not supported.

usSuspendFilter
USHORT — input

Not supported.

usIndexClass
USHORT — input

Not supported.

usNumCriteria
USHORT — input

The number of fields in the *pCriteria* array.

pCriteria
PLIBSEARCHCRITERIASTRUCT — input

The pointer to an array specifying the search criteria for each view you want to search. *pCriteria* 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 165.

ulMemListRequest
BOOL — input

This parameter controls how the search results are returned or which attribute values are returned. The VisualInfo for AS/400 system ignores this parameter if you set the *usStatDyn* parameter to SIM_SEARCH_BUILD_ONLY. Here are the valid values:

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. You can specify only one view for this option.

SIM_SEARCH_USER_SYSTEM_ATTR

Returns the item IDs, user attributes, and system attributes in a memory buffer. You can specify only one view for this option. Only the attributes in the AVT and SBTITEMS tables are returned.

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 169.

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 buffer. If you set the *usStatDyn* parameter to SIM_SEARCH_BUILD_ONLY, or if nothing matches the input search criteria, this field contains the value 0.

ulParam1
If you set the *ulMemListRequest* 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* or the number of items in the search results folder). The values in the *ulReturnLimit* field of the LIBSEARCHCRITERIASTRUCT structures limit this number.

If you set the *usStatDyn* parameter to SIM_SEARCH_BUILD_ONLY, or 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 B, "Guidelines for Search Expressions" on page 207.

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObjAcc
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.

ulOrigin
ULONG — input

The pointer origin index. The value of this parameter indicates the initial position of the object pointer. Here are the valid values:

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.

lOffset
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 *ulOrigin* parameter. This value can be either a negative or a positive byte count.

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 169.

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 *ulOffset*, 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.

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_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**

SimLibStoreNewObject (Store a New Object in an Existing Item)

Format

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

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObj
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 162.

ulConCls
ULONG — input

The content class identifier for the object (see Appendix F, "Predefined Content Classes" on page 220). 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 VisualInfo for AS/400 content class services to determine how to manipulate the contents of the objects you store.

pSMS
PSMS — input

Pointer to a system-managed storage (SMS) structure for an object. This structure uses only *szCollectionName*.

pObjBuffer
PVOID — input

The pointer to a memory buffer containing the object data.

ulObjSize
ULONG — input

The total size, in bytes, of the object.

lSeqAfterPart
LONG — input

Not supported.

ulAffiliatedType
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 VisualInfo for AS/400 system ignores this parameter. For more information on the ANNOTATIONSTRUCT structure, see “ANNOTATIONSTRUCT (Annotation Information Structure)” on page 152.

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 169.

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 VisualInfo for AS/400 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hItem
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.

usAttributeId
USHORT — input

The unique identifier assigned to an attribute.

pszAttributeValue
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 *usAttributeId* parameter.

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 169.

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_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 VisualInfo for AS/400 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 VisualInfo for AS/400-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* **)**

SimLibWriteObject

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.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hObjAcc
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.

pBuffer
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.

ulBytesToWrite
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.

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 169.

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 the number of bytes actually written.

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_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"                     /* VisualInfo for AS/400             */

 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**

SimWmCreateWorkPackage (Create a Work Package)

Format

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

Purpose

This function creates a new work package that an application can use for ad-hoc work control, allowing the application to route a work package containing a folder or document through one or more workbaskets without the requirement for a predefined process.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszWorkPackageDesc
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.

ulNumVariables
ULONG — input

Number of entries in the variable array. This field is ignored if the array *pVariableList* pointer is NULL.

pVariableList
PMVARSTRUCT — 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.

usWorkPriority
USHORT — input

Priority of the work to be performed. The priority affects the work sequencing as the work package moves through a process. A larger number is a higher priority. use a priority of zero to request the default priority.

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 169.

Return Values

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

usParam
Always zero.

ulParam1
Contains the work package ID.

ulParam2
Contains the work package instance.

ulRC
Contains one of the following return codes:

- 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**

SimWmEndProcess (End a Work Package on a Process)

Format

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

Purpose

This function forces an end to an active work package. It removes the work package from workbaskets.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 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

If only one instance exists, this parameter is ignored.

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 169.

Return Values

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

usParam
Not used.

ulParam1
Not used.

ulParam2
Not used.

ulRC
Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_PRIVILEGE_ERROR

Related Functions

- **SimWmCreateWorkPackage**
- **SimWmGetWorkPackage**

SimWmGetWorkBasketInfo (Get Information about a Workbasket)

Format

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

Purpose

Use this function to return information about the workbasket you specify.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszWorkBasketID
PSZ — input

Pointer to the name of the workbasket.

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 169.

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 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 181.

ulParam2
Not used.

ulRC
Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- 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

This function gets (opens) 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 highest priority work package currently available in the specified workbasket.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszWorkBasketID
PSZ — input

Pointer to the name of the workbasket.

ulWorkOrder
ULONG — input

Order used for selecting an entry from the workbasket. Here are the valid values. NULL is recommended to let the server determine the order.

SIMWM_ORDER_FIFO
Make selection based on first in, first out (FIFO) order.

SIMWM_ORDER_LIFO
Make selection based on last in, first out (LIFO) order.

SIMWM_ORDER_PRIORITY
Make selection 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 next work package.

ulInstanceID
ULONG — input

Identifier of the work package instance that distinguishes one parallel path from another within the process. Use zero to end all instances of the 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 169.

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 item and associated work management information.

ulParam2
Not used.

ulRC
Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_EMPTY_WORKBASKET
- SIM_RC_PRIVILEGE_ERROR

Guidelines for Use

Effects

- If the work package ID is not specified, this function will retrieve the next work package in the workbasket. Otherwise, the specified work package 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 this function to determine the priority assigned to a work package in a workbasket. 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 VisualInfo for AS/400 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. Specify zero to retrieve the next work package.

ulInstanceID
ULONG — input

Identifier of the work package instance that distinguishes one parallel path from another within the process. Use zero to end all instances of the 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 169.

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

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**

SimWmListWorkbaskets (List the Workbaskets)

Format

SimWmListWorkbaskets(*hSession, pAsyncCtl, pRC* **)**

Purpose

Use this function to get the names and IDs of all workbaskets defined in the system for which the user has authority.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 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 169.

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 ITEMNAMESTRUCT array.

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

ulRC
Contains one of the following return codes:

- SIM_RC_OK

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 **SimWmListWorkBasket** returns.

Follow-Up Tasks

When your application no longer needs the ITEMNAMESTRUCT array, use **SimLibFree** to free the buffer.

Related Functions

- **SimWmGetWorkBasketInfo**

SimWmQueryWorkPackage (Query a Work Package)

Format

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

Purpose

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

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 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.

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 169.

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 item and associated work management information.

ulParam2
Not used.

ulRC
Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_INVALID_INDEX_CLASS
- 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.

SimWmReturnWorkPackage (Return a Work Package to a Workbasket)

Format

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

Purpose

Use this 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 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.

usWorkPriority
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.

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 169.

Return Values

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

usParam
Not used.

ulParam1
Not used.

ulParam2
Not used.

ulRC
Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- 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. This function can also be used in combination with **SimWmGetWorkPackage** to obain information about a workbasket or to update data for a work package. **SimWmGetWorkPackage** opens the work package, and **SimWmReturnWorkPackage** closes the package, making again available in the workbasket.

Related Functions

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

SimWmRouteWorkPackage (Route a Work Package)

Format

SimWmRouteWorkPackage(*hSession, pszWorkBasketID, ulWorkPackageID, ulInstanceID, fRoute, pAsyncCtl, pRC* **)**

Purpose

Use this function to assign a work package to a workbasket. This function can be used to assign a work package (created with **SimWmCreateWorkPackage**) to a workbasket, or it can be used to reassign a work package from one workbasket to another.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pszWorkBasketID
PSZ — input

Pointer to the name of the workbasket.

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.

fRoute
BITS — input

Not supported.

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 169.

Return Values

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

usParam
Not used.

ulParam1
Not used.

ulParam2
Not used.

ulRC
Contains one of the following return codes:

- SIM_RC_OK
- SIM_RC_COMPLETION_ERROR
- SIM_RC_PRIVILEGE_ERROR

Related Functions

- **SimWmCreateWorkPackage**

SimWmSetWorkPackagePriority (Set the Priority of a Work Package)

Format

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

Purpose

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

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 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.

usPriority
ULONG — input

Not supported.

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 169.

Return Values

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

usParam
Always zero.

ulParam1
Contains the work package ID.

ulParam2
Contains the work package instance.

ulRC
Contains one of the following return codes:

- 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

Follow-Up Tasks

SimWmRouteWorkPackage should be called to route the work package to a workbasket. Use **SimLibLibFree(** *hSession, ulParam1, pRC***)** and **SimLibLibFree(** *hSession, ulParam2, pRC***)** to free the buffer.

Related Functions

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

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

Ip2CloseTOC

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hTOC
HTOC — input

The handle to the table of contents you want to close. Use the **SimLibGetTOC** function to get this handle.

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 169.

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_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**

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hTOC
HTOC — input

The handle to the table of contents that you want to refresh. Use the **SimLibGetTOC** function to get this handle.

usUpdate
USHORT — input

Not supported.

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 169.

Return Values

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

usParam
Contains the total number of items in the table of contents.

ulParam1
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 176.

ulParam2
Contains the handle to the table of contents.

ulRC
Contains one of the following return codes:

- 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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 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 169.

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
If the *ulParam2* 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 167.

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_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.

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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

usContentClassType
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.

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 169.

Return Values

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

usParam
Contains the value 1, to indicate that *ulParam1* contains a pointer. If no records exist for the specified content class type, this field contains the value 0.

ulParam1
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 161. If no records exist for the specified content class type, this field contains the value NULL.

ulParam2
Contains the number of content classes in the library server database. If *ulRC* contains an error code, *ulParam2* 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_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(** *ulParam1* **)** function to release allocated storage.

Ip2ListServers (List the Accessible Servers)

Format

Ip2ListServers(*pServrInfo, ulServrInfoSize, fSrchfilter, pRC* **)**

Ip2ListServers

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.

This function is supported only in the 32-bit Windows environment.

Parameters

pServrInfo
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.

ulServrInfoSize
ULONG — input

The size, in bytes, of the buffer allocated for the SERVERINFOSTRUCT array.

fSrchfilter
ULONG — 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 169.

Return Values

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

usParam
Contains the value 1.

ulParam1
If *usParam* 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 *ulServrInfoSize* parameter affects the value returned in *ulParam1*. For more information on the SERVERINFOSTRUCT data structure, see “SERVERINFOSTRUCT (Server Information Structure)” on page 171.

ulParam2
Contains the number of the servers returned by this call, though not necessarily the number of servers in the system.

ulRC
Contains one of the following return codes:

- 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 *ulServrInfoSize* 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.
- In these environments, in addition to returning a list of servers as defined in the client network table (FRNOLINT.TBL), this will also list the daemon FRNODAOS as a server. It is the responsibility of the application to filter this daemon from the list prior to exposing it to an end user.

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 **Ip2QueryQueryPrivBuffer** to determine access rights.

Parameters

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

usClassType
USHORT— input

Not supported.

usID
USHORT — input

The ID of an index class.

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 169.

Return Values

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

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

ulParam1
Contains a PSZ pointer. This pointer identifies the location of a CHAR *szPrivilege*[401] buffer where a data structure contains the evaluated privilege string.

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_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

pszPrivilege
PSZ — input

The current privileges set for the user.

ulAuthority
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.

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 169.

Return Values

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

usParam
Contains the value 1 if the privilege set represented by *pszPrivilege* contains the specified authority. Otherwise the field contains the value 0.

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_INVALID_POINTER
- SIM_RC_INVALID_PRC
- OIM_INVALID_PSZPRIVLEGE_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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

pitemidItem
PITEMID — input

The pointer to an item ID of a folder or workbasket.

usItemType
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.

usWipFilter
USHORT — input

Not supported.

usSuspendFilter
USHORT — input

Not supported.

usNbrOfClasses
USHORT — input

The number of index class identifiers in the list you specify as the value of the *pusClassIdList* parameter. Specify the value 0 for the *usNbrOfClasses* parameter to indicate that class is not a criterion for selecting items to count.

pusClassIdList
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 *usNbrOfClasses* parameter.

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 169.

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 the count of items in the table of contents. If no items satisfy the filtering criteria, this field contains the value 0.

ulParam2
Contains the value 0.

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_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

hSession
HSESSION — input

The handle to the VisualInfo for AS/400 session information. The **SimLibLogon** function creates the session information.

hTOC
HTOC — input

The handle to the table of contents for which you want to check the status. The **SimLibGetTOC** function returns this handle.

usCheck
USHORT — input

Not supported.

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 169.

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 6. Common Data Structures

This part provides more detailed reference information that describes the common data structures and database tables used for VisualInfo for AS/400. The data structures are listed alphabetically and are always in UPPERCASE in the VisualInfo for AS/400 code. The following information is provided about each data structure:

- Purpose
- Valid fields
- Valid field values
- Usage guidelines

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

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

AnnotationData
ANNOTATIONSTRUCT — output

The information associated with an annotation object. For more information, see ANNOTATIONSTRUCT (Annotation Information Structure).

ulObjType
ULONG — output

The type of object. Here are the valid values:

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.

Obj
OBJ — output

The object handle data structure that identifies the object. For more information, see HOBJ (Handle to Query Stored Object).

ulObjConCls
ULONG — output

The object content class of the object you query. The value SIM_CC_UNKNOWN indicates the undefined content class.

ulObjLength
ULONG — output

The length of the object in bytes.

lObjSeqAfter
LONG — output

The order of the object relative to other objects in the item.

Restriction: This is the value of the unsupported *lSeqAfterPart* parameter of the **SimLibCreateObject** function.

ulObjFlags
ULONG — output

Not supported.

tsCreate
TIMESTAMP — output

The date and time that the item or object was created.

tsChanged
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

ulStruct
ULONG — input/output

The length of the structure in bytes, including the length of this field.

ulPart
ULONG — input/output

The part number of the object. Only positive values are valid.

ulPageNumber
ULONG — input/output

The page number that the annotation object refers to.

usX
USHORT — input/output

The X coordinate for the annotation object on the page that the value of the *ulPageNumber* field references.

usY
USHORT — input/output

The Y coordinate for the annotation object on the page that the value of the *ulPageNumber* field references.

usT
USHORT — input/output

Not supported.

usAnnotUnused
USHORT — input/output

A reserved field.

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;
```

ATTRINFOSTRUCT

```
BOOL                       fSymmetricSwapping;
BOOL                       fShaping;
LONG                       lMin;
LONG                       lMax;
BITS                       fTypeFlags;
USHORT                     usAttrType;
USHORT                     usHorizontalOrientation;
USHORT                     usVerticalOrientation;
USHORT                     usMode;
USHORT                     usNumericSelectionDefault;
CHAR                       szAttributeName;
CHAR                       achLanguageCode;

} ATTRINFOSTRUCT, *PATTRINFOSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

fUseBidirectional
BOOL — output

Always set to FALSE.

fSymmetricSwapping
BOOL — input

This is always set to **FALSE**.

fShaping
BOOL — input

This is always set to **FALSE**.

lMin
LONG — input

The meaning of *lMin* varies with the value of the *usAttrType* parameter:

- It is the minimum length of the string and must contain the value 0 or a greater value, if *usAttrType* 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 *usAttrType* contains SIM_ATTR_LONG.

lMax
LONG — output

The meaning of *lMax* varies with the value of the *usAttrType* parameter:

- It is the maximum length of the string and must contain a value greater than 0 and greater than *lMin*, if *usAttrType* contains SIM_ATTR_FSTRING.

- It is the maximum value allowed if *usAttrType* contains SIM_ATTR_LONG.

fTypeFlags
BITS — output

Not supported.

usAttrType
USHORT — output

In VisualInfo for AS/400, this is always set to SIM_ATTR_VSTRING.

usHorizontalOrientation
USHORT — output

Not supported.

usVerticalOrientation—
USHORT — output

Not supported.

usMode
USHORT — output

Not supported.

usNumericSelectionDefault
USHORT — output

Not supported.

szAttributeName
CHAR[*SIM_ATTR_NAME_LENGTH*+1] — input/output

A NULL-terminated character string containing the application-defined name of the attribute.

achLanguageCode
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 *IBM National Language Design Guide: National Language Support Reference Manual Volume 2*.

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, which consists of the following:

```
typedef struct _ATTRLISTSTRUCT

{
ULONG                        ulStruct;
PSZ                          pszAttributeValue;
BITS                         fAttrFlags;
USHORT                       usAttrId;
```

ATTRLISTSTRUCT

```
USHORT                              usAttrType;

} ATTRLISTSTRUCT, *PATTRLISTSTRUCT;
```

Fields

ulStruct
ULONG — input/output

The length of the structure in bytes, including the length of this field.

pszAttributeValue
PSZ — input/output

The pointer to a NULL-terminated character string containing the value of an attribute.

fAttrFlags
BITS — output

Flags denoting 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. Here are the valid values. 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 VisualInfo for AS/400 system-defined attributes.

usAttrType
USHORT — input/output

In VisualInfo for AS/400, this is always set to SIM_ATTR_VSTRING.

VisualInfo for AS/400 supports the system-defined attributes shown in Table 1 on page 157.

Table 1. 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, *PCLASSATTRSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

fAttrRequiredField
BOOL — output

A flag that indicates whether a value is required for this attribute. Here are the valid values:

TRUE	Indicates that a value is required.
FALSE	Indicates that a value is not required.

Restriction: This field is valid for index classes only. It is not valid for index classes.

fAttrAccess
BITS — output

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. Here are the valid values:

SIM_ATTR_READABLE
Indicates read access.

SIM_ATTR_READWRITE
Indicates read and write access. This value is a combination of SIM_ATTR_READABLE and SIM_ATTR_WRITEABLE.

SIM_ATTR_WRITEABLE
Indicates write access.

usAttrId
USHORT — output

The unique identifier of an attribute.

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, *PCLASSINDEXATTRSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

usAttrId
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.

usIndexSortOrder
USHORT — output

In VisualInfo for AS/400, 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;
SZ                                  szIndexName;

} CLASSINDEXSTRUCT, *PCLASSINDEXSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

fIndexFlags
BITS — output

Not supported.

pClassIndexAttr
PCLASSINDEXATTRSTRUCT — output

A pointer to a ClassIndexAttrStruct data structure containing class index attribute information. For more information, see CLASSINDEXATTRSTRUCT (Class Index Attribute Structure).

usNbrAttrIds
USHORT — output

The number of attribute IDs in the ClassIndexAttrStruct structure.

szIndexName
SZ[*SIM_INDEX_NAME_LENGTH*+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;
```

CLASSINFOSTRUCT

```
USHORT                     usNbrAttrIds;
USHORT                     usMaxVersions;
USHORT                     usIndexClass;
USHORT                     usViewID;
CHAR                       szACLName;
CHAR                       achLanguageCode;
CHAR                       szClassName;
CHAR                       szDescription;
CHAR                       szCollectionName;
CHAR                       szStoreSite;

} CLASSINFOSTRUCT, *PCLASSINFOSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

pClassAttrStruct
PCLASSATTRSTRUCT — output

A pointer to an array of class attribute structures.

usNbrAttrIds
USHORT — output

The number of attribute IDs in the CLASSATTRSTRUCT array. For classes with no attributes, this value is 0, and the *pClassAttrStruct* field contains the value NULL.

usMaxVersions
USHORT — output

Not supported.

usIndexClass
USHORT — output

An index class identifier.

usViewID
USHORT — output

The ID of an existing index class view.

szACLName
CHAR[*SIM_ACCESS_LIST_NAME_LENGTH*+1] — output

Not supported.

achLanguageCode
CHAR[*SIM_LANGUAGE_CODE_LENGTH*+1] — output

The 3-character national language code for this index class name or view name. The values for language codes are described in the *IBM National Language Design Guide: National Language Support Reference Manual, Volume 2*.

szClassName
CHAR[*SIM_CLASS_NAME_LENGTH*+1] — output

The name of the index class or view, expressed in the specified language.

szDescription
CHAR[*SIM_DESCRIPTION_LENGTH*+1] — output

Not supported.

szCollectionName
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.

szStoreSite
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

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

usContentClsID
USHORT — output

An unique content class ID that VisualInfo for AS/400 generates.

szContentClsName
CHAR[9] — output

The name of the content class.

szContentClsDesc
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

ulStruct
ULONG — input/output

The length of the structure in bytes, including the length of this field.

ulPart
ULONG — input/output

The part number of the object. Only positive values are valid.

sVersion
SHORT — input

Not supported.

szItemID
ITEMID — input/output

The item ID of the object.

chRepType
UCHAR[*SIM_REP_TYPE*] — input/output

Not supported.

chReserved
UCHAR[*SIM_OBJ_RESERVED_LENGTH*] — 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;
PATTRLISTSTRUCT                pAttr;
USHORT                         usIndexClass;
USHORT                         usViewId;
USHORT                         usNumAttributes;

} ICVIEWSTRUCT, *PICVIEWSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

pNextView
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 this release of VisualInfo for AS/400, this pointer always contains the value NULL.

pAttr
PATTRLISTSTRUCT — output

The pointer to an array of ATTRLISTSTRUCT 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.

usIndexClass
USHORT — output

The index class identifier for the item.

usViewId
USHORT — output

The ID of an existing index class view.

VisualInfo for AS/400 supports only a single view, with the same identifier as the index class.

usNumAttributes
USHORT — output

The number of attribute values that exist for this item. The value of this field matches the number of ATTRLISTSTRUCT data structures that the *pAttr* 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;
```

ITEMINFOSTRUCT

```
BOOL                    fSuspended;
USHORT                  usItemType;
USHORT                  usIndexClass;
ULONG                   ulOpenStatus;
USHORT                  usWipStatus;
USERID                  useridCheckout;
CHAR                    szLabel;

} ITEMINFOSTRUCT, *PITEMINFOSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

fSuspended
BOOL — output

Not supported.

usItemType
USHORT — output

The type of items retrieved using the **SimLibGetItemInfo** function. Here are the valid values:

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.

usIndexClass
USHORT — output

An index class identifier.

For the **SimLibGetItemInfo** function, this value specifies the index class ID for the item you are querying.

ulOpenStatus
ULONG — output

Indicator of whether the item is open for update. Together, this parameter and the *useridCheckout* parameter provide information about who has the item and for what purpose. Here are the valid values:

SIM_ACCESS_READ_WRITE
Indicates that you have the item open for update.

SIM_ACCESS_UNKNOWN
Indicates that you do not have the item open for update.

usWipStatus
USHORT — output

Not supported.

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. Here are the valid values:

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.

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.

LIBSEARCHCRITERIASTRUCT

ulReturnLimit
ULONG — input

The maximum number of items that the search returns for the index classyou 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. Here are the valid values:

SIM_SEARCH_VIEW
Searches only the view specified in the *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 B, “Guidelines for Search Expressions” on page 207.

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

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

usID
USHORT — output

The ID of a valid attribute, an index class, or an index class view.

szName
CHAR[*SIM_CLASS_NAME_LENGTH*+1] — output

The name of the index class or view in the current language.

szDescription
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:

```
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;
```

OBJINFOSTRUCT

```
TIMESTAMP                     tsEnterSG;
TIMESTAMP                     tsEnterSC;
CHAR                          szCollectionName;
CHAR                          szObjectName;
CHAR                          szMgtCls;
CHAR                          szStgCls;
CHAR                          szDataCls;
CHAR                          szStoreSite;

} OBJINFOSTRUCT, *POBJINFOSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

ulObjSize
ULONG — output

The total size of the object in bytes.

lSMSRetention
LONG — output

Not supported.

lEstimateRetrieveTime
LONG — output

Not supported.

ulAvail
ULONG — output

Not supported.

ulObjConCls
ULONG — output

The object content class of the object you query. The value SIM_CC_UNKNOWN indicates the undefined content class.

usPageNum
USHORT — output

Not supported.

tsCreate
TIMESTAMP — output

The date and time that the item or object was created.

tsExpiration
TIMESTAMP — output

Not supported.

tsLastRef
TIMESTAMP — output

Not supported.

tsModify
TIMESTAMP — output

The date and time that the item or object was last modified.

tsEnterSG
TIMESTAMP — output

Not supported.

tsEnterSC
TIMESTAMP — output

Not supported.

szCollectionName
CHAR[*MAXCOLNMSZ*] — input

Not supported.

szObjectName
CHAR[*MAXOBJNMSZ*] — input

Not supported.

szMgtCls
CHAR[*MAXMGTCLSNMSZ*] — output

Not supported.

szStgCls
CHAR[*MAXSTGCLSNMSZ*] — output

Not supported.

szDataCls
CHAR[*MAXDATACLSNMSZ*] — output

Not supported.

szStoreSite
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;
```

RCSTRUCT

```
USHORT              usReserved;
USHORT              usParam;
ULONG               ulParam1;
ULONG               ulParam2;
ULONG               ulExtRC;
ULONG               ulExtReason;
PVOID               pApplData;
ULONG               ulApplData;
ULONG               ulReserved;
HERR                hErrLog;

} RCSTRUCT, *PRCSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

ulRC
ULONG — output

The function return code.

usReserved
USHORT — output

Not supported.

usParam
USHORT — output

A field that indicates whether the *ulParam1* field contains a pointer to a data area. The value 1 indicates that this is the case. Otherwise, this field contains the value 0.

ulParam1
ULONG — output

A value or a pointer to either a data structure or an array of data structures.

ulParam2
ULONG — output

A field that indicates the number of data structures in the array if the *ulParam1* field contains a pointer to an array of data structures.

ulExtRC
ULONG — output

A return code from other components that VisualInfo for AS/400 called directly or indirectly.

ulExtReason
ULONG — output

Not supported.

pApplData
PVOID — output

A PVOID data field that your application can use to contain application data. VisualInfo for AS/400 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. VisualInfo for AS/400 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 — output

The length of the structure in bytes, including the length of this field.

szServerName
CHAR[*SERVERNAME_LENG*+1] — output

The name of the VisualInfo for AS/400 server.

szServerType
CHAR[*SERVERTYPE_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

ulStruct
ULONG — input

The length of the structure in bytes, including the length of this field.

lSMSRetention
LONG — input

The period in days that VisualInfo for AS/400 retains the object in system-managed storage. The valid values range from 1 to 999 999 999.

szCollectionName
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.

szObjectName
CHAR[*MAXOBJNMSZ*] — input

Not supported.

szMgtCls
CHAR[*MAXMGTCLSNMSZ*] — input

Not supported.

szStgCls
CHAR[*MAXSTGCLSNMSZ*] — input

Not supported.

szDataCls
CHAR[*MAXDATACLSNMSZ*] — input

Not supported.

szStoreSite
CHAR[*MAXSTRSITENMSZ*] — input

The name of the object server in which the object is stored.

szStoreHint
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                           usNumWmSnapshots
USHORT                                      pWmSnapshot;
PICVIEWSTRUCT                               pICView;
USHORT                                      usNumViews;
USHORT                                      usItemType;
ULONG                                       ulOpenStatus;
ITEMID                                      szItemID;
USERID                                      useridCheckout;
TIMESTAMP                                   tsCreate;
TIMESTAMP                                   tsModify;

} SNAPSHOTSTRUCT, *PSNAPSHOTSTRUCT;
```

SNAPSHOTSTRUCT

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

pWmSnapshot
PWMSNAPSHOTSTRUCT — output

The pointer to the work management information data structure of the type WMSNAPSHORSTRUCT. The **SimLibGetItemSnapshot** function returns this structure when the you specify the value of the *fReadAttrInd* input parameter as SIM_WORK_ATTR. Otherwise, this field contains the value NULL.

VisualInfo for AS/400 supports the existence of an item in more than one workbasket, so it could be an array of work management information on an item.

usNumWmSnapshots
USHORT — input

The number of elements in the array of WMSNAPSHOTSTRUCT that *pWmSnapshot* points to.

pICView
PICVIEWSTRUCT — output

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.

Currently in VisualInfo for AS/400, 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.

usNumViews
USHORT — output

The number of elements in the linked list pointed to by the *pICView* field in the SnapshotStruct data structure.

Currently in VisualInfo for AS/400, 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 *pICView* pointer is still valid if you retrieve system attributes.

usItemType
USHORT — output

The type of items retrieved using the **SimLibGetItemSnapshot** function. Here are the valid values:

SIM_DOCUMENT
Indicates that the item is a document.

SIM_FOLDER
Indicates that the item is a folder.

ulOpenStatus
ULONG — output

Indicator of whether the item is open for update. Together, this parameter and the *useridCheckout* parameter provide information about who has the item and for what purpose. Here are the valid values:

SIM_ACCESS_READ_WRITE
Indicates that you have the item open for update.

SIM_ACCESS_UNKNOWN
Indicates that you do not have the item open for update.

szItemID
ITEMID — output

An item ID.

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.

tsCreate
TIMESTAMP — output

The date and time that the item or object was created.

tsModify
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

ulStruct
ULONG — input

The length of the structure in bytes, including the length of this field.

usItemStatus
USHORT — input

The status of the entry after the update. Here are the valid values:

- **0** (unmodified)
- **SIM_TOC_ADD**
- **SIM_TOC_MODIFIED**
- **SIM_TOC_DELETE**

usIndexClass
USHORT — input

An index class identifier.

usItemType
USHORT — input

The type of items retrieved using the **SimLibGetTOC** function. Here are the valid values:

SIM_DOCUMENT
Indicates that the item is a document.

SIM_FOLDER
Indicates that the item is a folder.

szItemID
ITEMID — input

An item ID.

tsItemChanged
TIMESTAMP — input

The timestamp of the item as stored in the library server.

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

ulStruct
ULONG — input

The length of the structure in bytes, including the length of this field.

ulUserType
ULONG — input

Not supported.

ulUserCCSID
ULONG — input

Not supported.

pszUserDescription
PSZ — input

Not supported.

szUserLanguage
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 *IBM National Language Design Guide: National Language Support Reference Manual Volume 2*

IBMszSessionType
CHAR[*SIM_SESSION_TYPE_LENGTH*+1] — input

The type of logon session. The only valid value for this field is lp2.

tsPasswordExpire
TIMESTAMP — input

The date when the current password expires.

szPrivString
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.

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

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

ulAccessLevel
ULONG — output

Not supported.

SIM_ACCESS_EXCL_READ
Not supported.

SIM_ACCESS_READ_WRITE
Opens the item or object for reading and writing. The object opens at the first byte of the object. Use of this value causes the open to fail if another process has opened the object for write access. This value opens the item or object for read access and write access.

useridCheckout
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.

szItemID
ITEMID — output

An item ID.

WMSNAPSHOTSTRUCT (Work Management Information Structure)

This data structure provides information about work management. It consists of the following:

```
typedef struct _WMSNAPSHOTSTRUCT

{
ULONG               ulStruct;
USHORT              usWIPStatus;
USHORT              usReleaseType;
USHORT              usPriority;
ITEMID              szWorkFlowID;
TIMESTAMP           tsWFEntry;
ULONG               ulWorkPackageID;
ULONG               ulInstanceID;
TIMESTAMP           tsEnteredWB;
ITEMID              szWorkBasketID;

} WMSNAPSHOTSTRUCT, *PWMSNAPSHOTSTRUCT;
```

Fields

ulStruct
ULONG — output

The length of the structure in bytes, including the length of this field.

usWIPStatus
USHORT — output

Not supported.

usReleasetype
USHORT — output

Not supported.

usPriority
USHORT — output

The current priority of the item within the workbasket.

szWorkFlowID
ITEMID — output

The workflow, if any, that this item is assigned to.

tsWFEntry
TIMESTAMP — output

The date and time when this item entered the listed workflow.

ulWorkPackageID
ULONG — output

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

ulInstanceID
ULONG — output

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

tsEnteredWB
TIMESTAMP — output

The date and time this item entered the listed workbasket.

szWorkBasketID
ITEMID — output

The ID of an existing workbasket.

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:

```
typedef struct _WMVARSTRUCT

{
CHAR                          szVarName;
CHAR                          szVarValue;

} WMVARSTRUCT, *PWMVARSTRUCT;
```

Fields

ulStruct
ULONG — input/output

The length of the structure in bytes, including the length of this field.

pszVarName
CHAR [SIMWM_VAR_NAME_LENGTH+1]— input/output

The name of the variable. The following constants represent the system variable names:

SIMWM_ITEMID
Items being routed.

SIMWM_INDEX_CLASS
Index class of the item.

SIMWM_PRIORITY
Priority of the work package.

SIMWM_FUNCTION
Identifier of the function or button selected by the user.

szVarValue
CHAR — 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                       szWorkBasketPrivString;

} WORKBASKETINFOSTRUCT, *PWORKBASKETINFOSTRUCT;
```

Fields

ulStruct
ULONG — input/output

The length of the structure in bytes, including the length of this field.

szWorkBasketName
CHAR[*OIM_WB_NAME_LENGTH*+1] — input/output

The name of the workbasket.

chAccessListName
CHAR[*ACCESS_LIST_NAME_SIZE*+1] — input/output

The name of access list for the workbasket.

WORKBASKETINFOSTRUCT

usWBLoadLimit
USHORT — input/output

Not supported.

bRemoveAfterIndex
BOOL — input/output

Not supported.

bSystemCntl
BOOL — input/output

A flag that indicates whether the system controls item priority within the workbasket. Here are the valid values:

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 of the work package and the order defined for the workbasket–LIFO, FIFO, or priority–determines the order.
FALSE	Indicates that this is not a system-assigned workbasket. The user can choose any item in the workbasket.

szUserFunName
CHAR[*OIM_WB_FUNCTION_LENGTH*+1] — input/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 *usWBLoadLimit* field. The DLL and function name are for use by your application. VisualInfo for AS/400 does not call this user exit.

szUserDLLName
CHAR[*OIM_WB_DLL_LENGTH*+1] — input/output

The name of a DLL that contains the user exit function. The DLL and function name are for use by your application. VisualInfo for AS/400 does not call this user exit.

szWorkBasketPrivString
UCHAR[*SIM_PRIVSTRING_LENGTH*+1] — input/output

The evaluated privilege string for the user with respect to the workbasket.

Chapter 7. 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.

Documents
The Documents property holds a collection of *Document* objects. A document, in Client for Windows terms is a Table of Contents view.

Error
The error information for the most recent method error.

Image
The Image property holds the VisualInfo for AS/400 document that is currently visible in the image viewer. If no document is visible, Image returns NULL.

Password
The Password property is the password to be used when the *Logon* method is called to log on to the VisualInfo for AS/400 library server. Refer to the description of the *Application* object's *Logon* method for a description of the possible values and results.

Server
The Server property contains the name of the VisualInfo for AS/400 library server that is logged on to when the Logon method is called. Refer to the description of the *Application* object's *Logon* method for a description of the possible values and results.

User
The *User* property contains the user ID that is used when the *Logon* method is called. Refer to the description of the *Application* object's *Logon* method for a description of the possible values and results.

Visible
The *Visible* property contains the visible status of the Windows Client frame window. The default value is False (0). This is a read/write property.

Data Type: VT_BOOL

Methods

The Application object supports the following methods.

ClassArray
The ClassArray method returns a safe array of VT_BSTRs containing the names of all of the VisualInfo for AS/400 index classes defined at the time the *Logon* method was executed. (;).

Arguments: None

Data Type: VT_VARIENT (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. The index classes are separated by semicolons (;).

Arguments: Index Class as VT_BSTR

Data Type: VT_VARIENT (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.

Arguments: IndexClass as VT_BSTR, Separator as VT_BSTR

Data Type: VT_BSTR

ClassList
The ClassList method returns a string with a list of all of the VisualInfo for AS/400 index classes defined at the time the *Logon* method was executed. The index classes are separated by the string separator argument.

Arguments: Separator as VT_BSTR

Data Type: VT_BSTR

ContentClassArray
The ContentClassArray 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.

Arguments: None

Data Type: VT_VARIENT (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.

Arguments: Separator as VT_BSTR

Data Type: 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.

Arguments: Source as VT_BSTR

Data Type: 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 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.

Arguments: Source as VT_BSTR

Data Type: VT_DISPATCH (Item)

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.

Arguments: Name as VT_BSTR

Data Type: VT_DISPATCH (Item)

ItemID
The *ItemID* method returns an *Item* object with the item ID specified. Refer to the *Item* object properties for a description of the *ItemID* property.

Arguments: *Item* as VT_BSTR

Data Type: VT_DISPATCH (Item)

KeyFieldArray
The KeyFieldArray method returns a safe array of VT_BSTRs containing the names of all of the VisualInfo for AS/400 index classes defined at the time the *Logon* method was executed. (;).

Arguments: None

Data Type: VT_VARIENT (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.

Arguments: Separator as VT_BSTR

Data Type: VT_BSTR

Logon
The *Logon* method logs on to VisualInfo for AS/400. 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 preinitialized 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 no-zero if there was an error.

Arguments: None

Data Type: VT_I4

OpenBasicSearch
The OpenBasicSearch method displays the basic search dialog box, allowing the operator to fill in a search. The resulting *Document* object is not returned.

Arguments: None

Data Type: VT_EMPTY

OpenScan
The OpenScan method displays the scan dialog box, allowing the user to select a scanner to open. Note that the resulting *Document* object is NOT returned.

Arguments: None

Data Type: VT_EMPTY

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.

Arguments: None

Data Type: VT_EMPTY

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.

Arguments: None

Data Type: VT_EMPTY

Search
The Search method returns an *Item* that represents the results of a search conducted on the VisualInfo for AS/400 file room with an optional index class and key field wildwood 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 165.

When TypeFilter=1, only folders are returned.

When TypeFilter=2, only documents are returned.

Any other TypeFilter value returns both documents and folders.

See Appendix B, “Guidelines for Search Expressions” on page 207 for search criteria.

Arguments:

- IndexClass as VT_BSTR (optional)
- SearchString as VT_BSTR (optional)
- TypeFilter as VT_VARIANT (optional, usually VT_I2)

Data Type: VT_DISPATCH (Item)

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.

Arguments: None

Data Type: VT_VARIENT

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.

Arguments: Separator as VT_BSTR

Data Type: VT_BSTR

Document Object

The *Document* object holds information about a Table of Contents (TOC).

Properties

Application
The Application property returns the Application object.

Count
The Count property returns the number of items that are listed in the TOC.

Item
The *Item* property returns the *Item* object that is associated with this *Document* (TOC).

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. This is a read/write property.

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).

SelectedCount
The SelectedCount property returns the number of items that are selected in the TOC.

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 as follows:

- 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.

Arguments: None

Data Type: 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.

Arguments: None

Data Type: VT_I4

ClearSelect
The ClearSelect method clears all of the current selections in the TOC.

Arguments: None

Data Type: 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.

Arguments: None

Data Type: 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.

Arguments: None

Data Type: 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.

Arguments: Page as VT_I4

Data Type: VT_I4

FirstPage

Displays the first page in a document. This method is valid only for documents, not workbaskets or folders.

Arguments: None

Data Type: 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.

Arguments: Index as VT_I4

Data Type: VT_DISPATCH (Items)

LastPage

Displays the last page in a document. This method is valid only for documents, not workbaskets or folders.

Arguments: None

Data Type: VT_I4

Maximize

The Maximize method maximizes the Document object in the main client window, hiding all other Document objects.

Arguments: None

Data Type: VT_I4

Minimize

The Minimize method minimizes the Document object in the main client window.

Arguments: None

Data Type: VT_I4

NextPage

Displays the next page (current page, plus 1) in a document. This method is valid only for documents, not workbaskets or folders.

Arguments: None

Data Type: VT_I4

PreviousPage

Displays the previous page (current page, minus 1) in a document. This method is valid only for documents, not workbaskets or folders.

Arguments: None

Data Type: VT_I4

Restore

The Restore method restores the Document object in the main client window to its original state (neither minimized or maximized).

Arguments: None

Data Type: VT_I4

Selections

The Selections method returns an *Items* collection containing all of the *Item* objects that are selected in the *Document* (TOC).

Arguments: None

Data Type: 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.

Arguments:

- First as VT_I4
- Last as VT_I4

Data Type: 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.

Arguments: VT_I4

Data Type: VT_I4

ZoomFit

The ZoomFit method lets you 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.

Arguments: None

Data Type: 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.

Arguments:

- Left as VT_I4
- Top as VT_I4
- Right as VT_I4
- Bottom as VT_I4

Data Type: VT_I4

Documents Object

The *Documents* collection object is a collection of all of the open *Document* objects (TOCs).

Properties

The *Documents* 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.

Count

The Count property holds the number of *Document* objects currently in the collection.

Parent

The Parent property returns the parent of the *Documents* collection object (which is the *Application* object).

Methods

The *Document* object supports the following 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.

Arguments: None

Data Type: VT_UNKNOWN.

Cascade

Cascade arranges all of the open Document objects that are not minimized in a cascaded manor.

Arguments: None.

Data Type: VT_I4

Close

The Close method closes all windows associated with the *Documents* objects and removes the Document objects from the *Documents* collection.

Arguments: None

Data Type: VT_EMPTY

CloseIt

NOTE: 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.

Arguments: None

Data Type: VT_I4

Item

The *Item* method returns one of the *Document* objects contained in the collection.

Arguments: Index as VT_I4

Data Type: 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.

Arguments:

- Index as VT_DISPATCH (Item)
- Browse as VT_VARIANT (optional, usually VT_BOOL)

Data Type: VT_DISPATCH (Document)

OpenTOC

The *OpenTOC* method creates a new *Document* object and adds it to the *Documents* collection.

Arguments: Index as VT_DISPATCH (Item)

Data Type: VT_DISPATCH (Document)

Tile

The Tile method arranges all of the open Document objects that are not minimized in a tiled manor. The Vertical argument specifies if the objects should be set primarily vertically (nonzero) or horizontally (zero).

Arguments: Vertical as VT_I4

Data Type: VT_I4

Error Object

The *Error* object describes the details about any error that may have happened while executing a method in the 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 the Client for Windows was doing at the time.

ExtReturnCode
The ExtReturnCode property contains the VisualInfo for AS/400 extended return code that was returned when the error was detected.

ReturnCode
The ReturnCode property contains the VisualInfo for AS/400 error code that was returned when the error was detected.

Methods

The *Error* object does not have any methods.

Image Object

The *Image* object holds the currently visible VisualInfo for AS/400document.

Properties

The *Image* object supports the following properties.

Application
The *Application* property returns the Application object.

Item
The *Item* property returns the *Item* object that is associated with this Image.

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. This is a read-write property.

Data Type: VT_I4

Parent
The Parent property returns the parent of the *Image* object (which is the *Application* object).

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.

Arguments: Save as VT_BOOL

Data Type: VT_EMPTY

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 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.

Arguments: Save as VT_BOOL

Data Type: VT_EMPTY

DisplayPage

The DisplayPage method forces the page specified to be displayed in the image viewer.

Arguments: Page as VT_I4

Data Type: VT_I4

FirstPage

The FirstPage displays the first page in the viewer.

Arguments: None

Data Type: VT_I4

LastPage

The LastPage displays the last page in the viewer.

Arguments: None

Data Type: VT_I4

NextPage

The NextPage displays the next page (current page + 1) in the viewer.

Arguments: None

Data Type: VT_I4

OpenDocument

The *OpenDocument* method opens a new VisualInfo for AS/400 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.

Arguments: Index as VT_DISPATCH (Item)

Data Type: VT_I4

PreviousPage
The PreviousPage displays the previos page (current page – 1) in the viewer.

Arguments: None

Data Type: VT_I4

Item Object

The *Item* object represents a VisualInfo for AS/400 item like a folder, workbasket, or document.

Properties

The *Item* object supports the following properties.

Application
The Application property returns the Application object.

CheckedStatus
The CheckedStatus property returns the user who has the item checked out, if any.

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.

ItemID
The *Item*ID is a string that uniquely defines each item in the VisualInfo for AS/400 fileroom. This string is a 12-character alphanumeric string that can be used as an index into a database if desired.

KeyFields
The KeyFields property is an array containing the values of all the key fields. KeyFields has one argument: the name of the key field. Following is an example of how you can use the KeyFields property in VisualBasic:

```
Dim Folder as Object

...

count = Folder.KeyFields("UseCount")   ' Get value
count = count + 2
Folder.KeyFields("UseCount") = count   ' Set value
Folder.UpdateIndex
```

When you set the index class, you must call the UpdateIndex method to confirm the change. This is a read/write property.

Data Type: VT_BSTR

Name

The Name property returns VisualInfo for AS/400'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.

PartCount

The PartCount property returns the number of parts stored in a document.

Parent

The Parent property returns the parent of the *Image* object (which can be the *Application* object or an *Items* object).

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, *Priority* returns the class default priority, and is read-only.

SystemAssigned

Returns TRUE if the workbasket is a system-assigned workbasket.

TOCCount

The TOCCount property returns the number of items that are indexed in this table of contents.

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.

Methods

The *Item* object supports the following methods.

Activate

The Activate method removes the suspended status from a suspended item.

Arguments: None

Data Type: 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.

Arguments:

- Path as VT_BSTR
- ContentClass as VT_I4

Data Type: VT_I4

AddToFolder

The AddToFolder method adds the *Item* to the folder specified as another *Item* object.

Arguments: Folder as VT_DISPATCH (Item)

Data Type: VT_I4

ChangeNotes

The ChangeNotes method saves the string passed in as the new note log. If you want to append to the note log, make sure you pass the whole string, not just the new part.

Arguments: Notes as VT_BSTR

Data Type: VT_I4

CheckIn

The CheckIn method checks the item in, letting anyone modify it.

Arguments: None

Data Type: VT_I4

CheckOut

The CheckOut method checks the item out to the current user, disabling anyone else from modifying it.

Arguments: None

Data Type: VT_I4

Close

The Close method unlocks the item previously locked with the Open method or NextWorkbasketItem (the resulting item, not the workbasket).

Arguments: None

Data Type: 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 unlocks the item previously locked with the Open method or NextWorkbasketItem (the resulting item, not the workbasket).

Arguments: None

Data Type: VT_I4

CloseNotes

The CloseNotes method unlocks the note log so other clients can open it. No changes are saved. See the ChangeNotes method to save changes.

Arguments: None

Data Type: VT_I4

CloseParts

The CloseParts method closes all of the open part files (pages) without saving any changes.

Arguments: None

Data Type: VT_I4

Delete

The Delete method removes the item from the file room. This is a nonrecoverable operation, so use this method with care.

Arguments: None

Data Type: VT_I4

DeletePart

The DeletePart method deletes the specified object (part) from the item.

Arguments: Index as VT_I4

Data Type: 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 (nonzero), enables you to fax the documents contained in folders.

Arguments: withSubFolderContents as VT_VARIANT (optional, usually VT_BOOL)

Data Type: VT_I4

GetNotes

The GetNotes method returns the text of the note log for the item or a blank string if none exists.

Arguments: None

Data Type: VT_BSTR

GetPartContentClass

The GetPartContentClass method returns the content class name of the part.

Arguments: Index as VT_I4

Data Type: VT_BSTR

GetPartFile

The GetPartFile method retrieves a part file from VisualInfo for AS/400, stores it in a temporary file on the local workstation, and returns the full path to the temporary file. The *Item* is checked out in VisualInfo for AS/400 the first time you call this method.

Arguments: Index as VT_I4

Data Type: BSTR

GetTOCItem

The GetTOCItem method returns the *Item* object specified from the TOC.

Arguments: Index as VT_I4

Data Type: VT_DISPATCH (Item)

NextWorkbasketItem

The NextWorkbasketItem method returns the next available item by order of priority in a workbasket.

Arguments: None

Data Type: 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.

Arguments: None

Data Type: None

PrintItem

The PrintItem method sends the item to the printer. If the Setup argument is set to TRUE (nonzero), the print setup dialog is displayed to the user first, allowing the user to specify print options.

Arguments: Setup as VT_BOOL

Data Type: VT_I4

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.

Arguments: None

Data Type: VT_I4

RemoveFromFolder

The RemoveFromFolder method removes the item from the folder specified as an argument.

Arguments: Workbasket as VT_DISPATCH (Item)

Data Type: VT_I4

RemoveFromWorkbasket

The RemoveFromWorkbasket method removes the item from the workbasket specified as an argument.

Arguments: Workbasket as VT_DISPATCH (Item)

Data Type: 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.

Arguments:

- Workbasket as VT_DISPATCH (Item)
- Priority as VT_VARIANT (optional, usually VT_I4)
- Force as VT_VARIANT (optional, usually VT_BOOL)

Data Type: VT_I4

SavePart

The SavePart method saves any changes that occurred to the part file specified and its annotation file.

Arguments: Index as VT_I4

Data Type: VT_I4

StartWorkflow

The StartWorkflow method adds the item into the specified workflow.

Arguments:

- Workflow as VT_BSTR
- Initial Workbasket as VT_DISPATCH (Item) (optional)
- Priority as VT_I4 (optional)

Data Type: 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.

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:

```
1995-04-01-08.05.23.000000
```

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.

Arguments:

- Timestamp as VT_BSTR
- TimeoutWorkbasket as VT_DISPATCH
- Classes as VT_BSTR
- Criteria as VT_I4
- ReadyWorkbasket as VT_DISPATCH

Data Type: VT_I4

UpdateIndex

The UpdateIndex method saves any changes that you have made to the Index Class and/or key fields (using the Properties Class and KeyFields). Until this method is called no changes are stored.

Arguments: None

Data Type: 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.

Count

The Count property returns the number of *Item* objects referenced in the *Items* collection.

Parent

The Parent property returns the parent of the *Items* collection (which is usually a *Document* object).

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.

Arguments: None

Data Type: VT_UNKNOWN

Close

The Close method closes the *Items* collection.

Arguments: None

Data Type: VT_I4

CloseIt

NOTE: 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.

Arguments: None

Data Type: VT_I4

Item

The *Item* method returns an *Item* object from the *Items* collection.

Arguments: Index as VT_I4

Data Type: VT_DISPATCH (Item)

Appendixes

Appendix A. VisualInfo for AS/400 Terminology

This guide uses the following terms to help you program applications using VisualInfo for AS/400. You can use these definitions as an introduction to VisualInfo for AS/400 concepts and then as a reference as you create applications.

General Terms

The following terms are organized by VisualInfo for AS/400 component.

VisualInfo for AS/400

The following terms are related to VisualInfo for AS/400.

Attribute
Characteristic associated with items in an index class.

Content Class
Describes the physical format of an object stored in the object server. Applications use it to launch an appropriate application or correctly display data.

Data Model
Provides a logical view of the organization of data in a database. The VisualInfo for AS/400 data model provides your applications with many general document and folder management capabilities.

Document
A type of item in VisualInfo for AS/400 and a basic part of the VisualInfo for AS/400 data model. It is similar to a paper document. In VisualInfo for AS/400, *document* can be any multimedia, digital object.

Folder
A type of item and a basic part of the VisualInfo for AS/400 data model that is similar to folders in a paper filing system and can contain other folders or documents.

Index Class
Specifies attributes common to a set of items. For example, an index class for reports could have attributes such as author, data, or subject. When you create an item, your application must assign an index class and supply the attribute values required by that class. You can create separate index classes for reports, spreadsheets, claim forms, or other kinds of documents that you use.

Item
An independent entity in the VisualInfo for AS/400 library server. It can be a document or a folder. The VisualInfo for AS/400 APIs let your application create, index, and locate items.

Item Part
Associated with an item and contains content such as image data, annotations, or notes.

Note
Text that the user enters in the VisualInfo for AS/400 client to describe the document or folder, to record actions taken, or communicate with others when they access the item.

Privilege
Capability that the system administrator gives to a user to either access or perform certain tasks on objects stored in the system.

Workbasket
A container that holds work packages–which can refer to folders and documents–to be processed in a predefine order.

Client Application

The following terms are related to the VisualInfo for AS/400 client.

Advanced Search Settings Notebook
Lets you create and modify existing advance search profiles. You see an advanced search setting notebook by selecting the New Search Profile icon in the Fileroom Search container, or by selecting Open, and then Settings, when an existing search template is highlighted. When you select an existing advanced search profile from the Fileroom Search container, a modified Advanced Search notebook appears that lets you enter new data values to use as search criteria for the advanced search.

Basic Search Dialog
Lets you enter simple search criteria associated with a single index class or all classes. You see a Basic Search dialog by selecting Basic search from the Tools menu in the main container.

Export Dialog
Lets you specify user options for exporting documents and folders from VisualInfo for AS/400 to files. You see the Export dialog by selecting Export from the Document, Folder, and Selected menus.

Image Window
Can display a document or an icon representing an object of an externally supported content class.

Import Dialog
Lets you select files to include in VisualInfo for AS/400. You see an Import dialog by selecting the Import option from the File menu.

Index Form Window
The Index window contains the current values for the key fields (attributes) associated with the displayed item and is used to specify a new index class and attribute values.

Logon Dialog
Appears when you start the client application. You enter a user ID and password and select the desired library server for the session in the Logon dialog.

Main Container
The *main container* displays icons representing other containers and a menu bar that lets you select tools and session-wide options. After logon, you see the main container.

Note Log Window
The Note Log window contains user-written notes associated with the displayed item. The Note Log window is implemented as an edit window, allowing easy input of text.

Print Dialog
Lets you specify user options for the printer profile you want to use, pages of documents you want to print, and parts of documents and folders that you want to print.

Scanner Windows
Lets you specify scanner operational controls. You see a scanner window by selecting the Basic scan or Advanced scan options from the File menu in the main container. A View window also might appear, based on user preferences, when the scanner window appears. The format of the scanner window can vary depending upon the scanner used.

Table of Contents Window
The table of Contents window displays a list of the contents in a workbasket or a folder.

Workbaskets Container
Has entries for all workbaskets for which you have read access.

Object Server

The following terms are related to an object server.

Collection
A *collection* is a group of objects with a similar set of storage management rules.

Object
An *object* is a stored stream of bits. An object can be an image, spreadsheet, word processing file, note, annotation, or event associated with a document. See *part*.

System-Managed Storage
System-managed storage (SMS) is an approach to efficient storage management in which the system determines document placement and an automatic data manager handles data, movement, space, and security.

Appendix B. Guidelines for Search Expressions

Included in this appendix are some guidelines to follow when you are searching a VisualInfo for AS/400 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:

```
Annn
```

Where the fields have the following meanings:

A	An attribute. You can enter attributes in uppercase, lowercase, or mixed case.
nnn	A decimal attribute ID. This value identifies either a user-defined attribute or a system-defined attribute as it exists in VisualInfo for AS/400.

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.

Appendix C. Using FlowMark with VisualInfo for AS/400

The following information describes how to integrate FlowMark with VisualInfo for AS/400. Currently, the level of integration is based on the use of the OLE automation interface of the Client for Windows. This integration is enhanced through the VisualInfo for AS/400 client high-level programming interface for Visual Basic, and the Windows executable, FRNWWFFM.EXE, which enables VHLPI functions to be called from FlowMark and sets FlowMark container variables.

VisualInfo for AS/400 Client High-Level Programming Interface for Visual Basic

The VisualInfo for AS/400 client high-level programming interface for Visual Basic is composed of the Visual Basic code module, FRNWWFVB.BAS. You can load it into any Visual Basic program to provide high-level function calls to the Client for Windows's OLE automation interface.

You can create Visual Basic applications that include the FRNWWFVB.BAS code module to call one or more VisualInfo for AS/400 functions. Table 2 lists the functions available with the syntax and return information.

You can write FlowMark activity programs to use one or more VHLPI functions. In addition, these programs can be designed to read and write FlowMark container variables. Using container variables as input/output for VHLPI functions results in a very tight integration between FlowMark and VisualInfo for AS/400. This level of integration is also provided by the FRNWWFFM.EXE program included with VisualInfo for AS/400.

Table 2 (Page 1 of 2). Visual Basic Function Summary

Function Name	Purpose
VbVhlAddFolderItem	Adds an item to a folder.
VbVhlAdminItemNoteLog	Reads, appends, replaces or deletes note logs.
VbVhlChangeItemIndex	Changes the index class of an item to another index class.
VbVhlCheckInItem	Unlocks the item.
VbVhlCheckOutItem	Locks the item for exclusive update access.
VbVhlCloseDocViews	Closes the document image window.
VbVhlCopyDoc	Creates a document and copies the contents of an existing document into it.
VbVhlCreateFolder	Creates a new folder.
VbVhlCreateFolderAddItem	Creates a folder and adds the specified item into it.
VbVhlDeleteItem	Deletes the item from VisualInfo for AS/400.
VbVhlDisplayDocView	Displays a document image using the Image viewer.
VbVhlDisplayVIItem	Displays a document, folder, or workbasket through the Client for Windows.
VbVhlDropFuncs	End access to VHLPI functions.
VbVhlExportDocObj	Creates an external file from a document base object.
VbVhlGetVIUserID	Returns the user ID logged onto VisualInfo for AS/400.
VbVhlImportDocObj	Creates a document base object from an external file image.

Table 2 (Page 2 of 2). Visual Basic Function Summary

Function Name	Purpose
VbVhlListContClasses	Lists all content classes.
VbVhlListFolderItems	Lists all folder items (of specified index classes).
VbVhlListFolderItemsAttr	Lists all folder items and their attribute values.
VbVhlListIndexClassAttr	Lists all attributes of a specified index class.
VbVhlListIndexClasses	Lists all index class names.
VbVhlListItemCC	Lists the content class of an item's base object.
VbVhlListItemInfo	Lists an item's type, index class name, index attributes and values.
VbVhlListWBItems	Lists all items in a specified workbasket.
VbVhlListWorkBaskets	Lists all workbasket names.
VbVhlLoadFuncs	Get access to VHLPI functions.
VbVhlLogoff	End access to VisualInfo for AS/400.
VbVhlLogon	Get access to VisualInfo for AS/400.
VbVhlRemoveFolderItem	Deletes the item only from the specified folder.
VbVhlScanDoc	Calls the VisualInfo for AS/400 scan facility.
VbVhlSearchAdv	Returns all items which match the advanced search criteria.
VbVhlSearchItem	Returns all items which match the index class and index attribute specification.

Using VHLPI Functions Through the FlowMark Command Line

FlowMark activities use registered programs to execute. Registered program information consists of the execution characteristics of the program, including the name of a program and any input parameters passed to the program as command line parameters. These input parameters can be FlowMark container variables if the parameter is registered with % characters surrounding its name, for example *%variable1%*.

FRNWWFFM.EXE is a Windows executable which enables VHLPI Visual Basic functions to be called from FlowMark activities. To accomplish this, the FlowMark program registration would have FRNWWFFM.EXE defined as the program name with an input parameter list containing the VHLPI Visual Basic function name followed by any parameters used by the function. For example, to display a document whose item ID is SSIINIR$D2G#V@04, the following would be the program registration parameters list:

```
VbVhlDisplayDocView,SSIINIR$D2G#V@04,False
```

See Table 2 on page 211 for the list of VHLPI Visual Basic functions which can be called by FRNWWFFM.EXE. FlowMark container variables can be used as parameters. For example if the FlowMark container variable *ItemId* was set to the value SSIINIR$D2G#V@04, then the following parameter list could be used:

```
VbVhlDisplayDocView,%ItemId%,False
```

When the VHLPI function calls for an array, each element of the array must be included as a parameter. The first element of every array contains the element count of the array. If a FlowMark container variable name is used, then the container variable must also be an array. For example:

```
VbVhlImportDocObj,ItemId,c:\test.txt,TEXT,NOINDEX,1,Source,1,IMPORT
VbVhlImportDocObj,ItemId,c:\test.txt,TEXT,NOINDEX,%AttrName%,%AttrValue%
```

In addition to providing FlowMark command line access to VHLPI functions, FRNWWFFM.EXE also copies all FlowMark input container variables into corresponding output container variables. This ensures that subsequent FlowMark activities using the output container from FRNWWFFM.EXE will have properly set container values. This is necessary because FRNWWFFM.EXE provides a method to set output container variables specified on the program registration parameter list.

To have FRNWWFFM.EXE set FlowMark output data container variables, the variable name must be registered on the program parameter list prefixed with an OUT. This only applies to VHLPI function parameters that contain return data. This lets you specify an output container variable to receive the output data returned by VHLPI functions. Subsequent activities can use this data for other processing.

For example, you can use FRNWWFFM.EXE in two sequential activities to first import a document and then display the document. In FlowMark, two programs would be registered both using FRNWWFFM.EXE as the program name. The two programs could have the following parameter lists:

```
VbVhlImportDocObj,OUT.ItemId,c:\test.txt,TEXT,NOINDEX,0,0
VbVhlDisplayDocView,%ItemId%,False
```

The first program imports a document into the NOINDEX index class and returns the item ID to the FlowMark output container variable *ItemId*. By defining the output container of the first program to be used as input to the second program, the second program displays the document whose item ID is read from the input container variable *ItemId*.

If the output parameter is an array, the FlowMark container variable must be defined as an array and be large enough to hold all the output.

Summary

Because FRNWWFFM.EXE sets FlowMark containers, it can only be executed as an FlowMark program. This capability enables VHLPI functions to be called from FlowMark without the need to create a VisualBasic program. VHLPI functions, enabled by the Client for Windows's OLE automation interface, provide FlowMark designers with the capability of creating workflow models which access VisualInfo for AS/400 function without coding customized programs.

Appendix D. Data Structures and Definitions

This appendix describes the abstract data types, data structures, and definitions the VisualInfo for AS/400 client high-level programming interface uses.

ADMINITEMNOTELOGOUT

Structure

The structure for VhlAdminItemNoteLog() output data.

```
typedef struct _AdminItemNoteLogOut {
   ULONG      ulNoteSize;          /* NoteLog text size                   */
   CHAR       achNoteLogText[1];   /* actual NoteLog text                 */
}  ADMINITEMNOTELOGOUT;
typedef ADMINITEMNOTELOGOUT *PADMINITEMNOTELOGOUT;
```

ATTRBPAIR

Structure

The structure for Index Class - attribute name and value.

```
typedef struct AttributePair {
   PSZ        pszAttrbName;        /* Pointer to Attribute Name           */
   PSZ        pszAttrbValue;       /* Pointer to Attribute Value          */
}  ATTRBPAIR;
typedef ATTRBPAIR *PATTRBPAIR;
```

INSTANCSTRUCT

Structure

The structure containing Instance (Service) data. This is used for adding user defined functions with the Service Broker Manager.

```
typedef struct InstanceDataStructure {
   CHAR            szService[9];   /* SBM Service name                    */
   CHAR            szBroker[9];    /* SBM Broker name                     */
}  INSTANCSTRUCT;
typedef INSTANCSTRUCT *PINSTANCSTRUCT;
```

ITEMATTRB

Structure

The structure for Item Id with attribute names and values.

```
typedef struct ItemAttributes {
   ITEMID        pszItemID;         /* Item Id of item                      */
   USHORT        usNumAttrb;        /* Numb of attribute name/values in list */
   PATTRBPAIR    pAttrList;         /* Attribute name/value list            */
}  ITEMATTRB;
typedef ITEMATTRB *PITEMATTRB;
```

LFOLDERTOCDATA

Structure

The structure for VhlListFolderItems() output data.

```
typedef struct ListFolderTOCOutData {
   ITEMID        szItemID;                              /* Item ID       */
   USHORT        usItemType;                            /* Item Type     */
   CHAR          szClassName[SIM_CLASS_NAME_LENGTH+1];  /* Index Class   */
   PVOID         pUserStruct;                           /* Spare-NOT USED */
}  LFOLDERTOCDATA;
typedef LFOLDERTOCDATA *PLFOLDERTOCDATA;
```

LISTCCINFODATA

Structure

The structure for Content Class information output data.

```
typedef struct ListCCInfoData {
   CHAR          szName[9];                         /* Content Class name */
   CHAR          szDesc[41];                        /* Content Class desc */
   PVOID         pUserStruct;                       /* Spare - NOT USED   */
}  LISTCCINFODATA;
typedef LISTCCINFODATA *PLISTCCINFODATA;
```

LISTFOLDERITEMATTRSTRUCT

Structure

The structure for VhlListFolderItemsAttr() output data.

```
typedef struct _ListFolderItemAttrData {
   USHORT             usNbrItemIDs;              /* #ItemIDs in folder    */
   SBVIITEMINFOSTRUCT aItemInfoStruct[1];        /* ptr to ItemInfo       */
}  LISTFOLDERITEMATTRSTRUCT;
typedef LISTFOLDERITEMATTRSTRUCT *PLISTFOLDERITEMATTRSTRUCT;
```

LISTICATTRDATA

Structure

The structure for Index Class attribute data.

```
typedef struct ListICAttrData {
   CHAR         szAttrName[SIM_ATTR_NAME_LENGTH+1];   /* Attribute name     */
   BOOL         fRequired;                            /* Required field  ?? */
   BOOL         fIndexKeyField;                       /* Index Key Field ?? */
   BITS         fIndexKeyUnique;                      /* Index Key Unique?? */
   BITS         fTypeFlags;                           /* Data type flags    */
   USHORT       usAttrType;                           /* Attribute type     */
   LONG         lMin;                                 /* Min values         */
   LONG         lMax;                                 /* Max values         */
   PVOID        pUserStruct;                          /* Spare - NOT USED   */
}  LISTICATTRDATA;
typedef LISTICATTRDATA *PLISTICATTRDATA;
```

LWBASKETDATA

Structure

The structure for VhlListWorkBaskets() output data.

```
typedef struct ListWorkBasketOutData {
   ITEMID       szItemID;                               /* Item ID         */
   CHAR         szWBasketName[OIM_ITEMNAME_LENGTH+1];   /* WorkBasket name  */
   PVOID        pUserStruct;                            /* Spare - NOT USED */
}  LWBASKETDATA;
typedef LWBASKETDATA *PLWBASKETDATA;
```

SBVIITEMINFOSTRUCT

Structure

The structure for VhlListItemInfo() output data.

```
typedef struct _ItemInfoStruct {
   ITEMID     szItemID;                          /* ItemID                 */
   USHORT     usItemType;                        /* Doc‘Folder‘Wbasket     */
   CHAR       szClassName[SIM_CLASS_NAME_LENGTH+1];/* Index class for search */
   USHORT     usNbrAttrIds;                      /* input # of ret items   */
   PATTRBPAIR pAttrList;                         /* ptr to Attr pair struct*/
   PVOID      pUserStruct;                       /* Spare - NOT USED       */
} SBVIITEMINFOSTRUCT;
typedef SBVIITEMINFOSTRUCT *PSBVIITEMINFOSTRUCT;
```

Appendix E. Return Codes

The VisualInfo for AS/400 client high-level programming interface returns these types of errors:

- Internal VHLPI function errors
- Other VisualInfo for AS/400 component errors

Descriptions of internal VHLPI function errors are given in the following sections. For information on other component errors, refer to the *IBM ImagePlus VisualInfo: Messages and Codes*.

Internal VHLPI Return Codes

Table 3 lists the error codes generated by the internal processing of the VisualInfo for AS/400 high-level programming interface.

Table 3 (Page 1 of 3). Error Codes from the VisualInfo for AS/400 High-Level Programming Interface

Error Code	Message Text	Explanation
900	SBVI_FAILED	A general failure of VisualInfo for AS/400 occurred.
901	SBVI_ILLEGAL_CLASSNAME	The specified class name is not valid.
902	SBVI_INVALID_PARAMS	An incorrect number or type of parameters was specified.
903	SBVI_TOO_MANY_PARAMS	The specified parameters were too large for the stack.
904	SBVI_REXX_VAR_ERROR	An error occurred when trying to set the REXX variable value.
905	SBVI_BAD_INDXCLS	The specified index class does not exist or duplicates an existing index class.
906	SBVI_BAD_ATTRIBUTE	The specified attribute does not exist or duplicates an existing attribute.
907	SBVI_BAD_ITEMID	The item ID does not exist or was not valid.
908	SBVI_BAD_FOLDER	The folder ID does not exist or was not valid.
909	SBVI_BAD_DOCUMENT	The document ID does not exist or was not valid.
910	SBVI_BAD_PARAM_VALUE	The specified parameter value was not valid.
911	SBVI_OPEN_FILE_ERROR	An error occurred when trying to open the file.

Table 3 (Page 2 of 3). Error Codes from the VisualInfo for AS/400 High-Level Programming Interface

Error Code	Message Text	Explanation
912	SBVI_READ_FILE_ERROR	An error occurred when trying to read the file.
913	SBVI_WRITE_FILE_ERROR	An error occurred when trying to write to the file.
914	SBVI_BUFFER_OVERFLOW	The buffer was too small to store data.
915	SBVI_PM_ERROR	A Presentation Manager error occurred.
916	SBVI_MEMORY_ERROR	The needed memory could not be allocated.
917	SBVI_SEARCH_NO_MATCH	No items were found that matched the search criteria.
918	SBVI_INVALID_REXX_VAR	The specified REXX variable name was not valid.
919	SBVI_EMPTY_LIST	No items were returned.
920	SBVI_REQ_OUTPUT_TOO_BIG	The required output data exceeds the maximum allowed.
921	SBVI_UNSUPPORTED_CC	The specified content class is not supported.
922	SBVI_ILLEGAL_FILENAME	The specified filename is not valid.
923	SBVI_ILLEGAL_CC	The specified content class is not valid.
924	SBVI_ITEM_CHECKEDOUT	The item or parent folder is locked by another user.
925	SBVI_BAD_SNAPSHOT	The snapshot information that was obtained is damaged.
926	SBVI_ITEM_NOT_CHECKEDOUT	The item is not yet locked.
927	SBVI_UPDATE_NOT_ALLOWED	No update access is allowed.
928	SBVI_OPEN_NOTELOG_ERROR	The note log could not be opened.
929	SBVI_ZERO_BUFFER_SIZE	The specified buffer size was invalid.
930	SBVI_MAXFILEHANDLE_ERROR	The maximum number of file handles could not be set.
931	SBVI_EMPTY_DOCUMENT	The document contains no pages.
932	SBVI_INVALID_FLAG	The flag was not Y/N and so was not valid.
933	SBVI_REQ_ATTR_MISSING	The required attributes are missing.
934	SBVI_BAD_WORKBASKET	The workbasket is either not defined or not valid.
935	SBVI_INVALID_CLASSNAME	The index class name is not valid.
936	SBVI_NO_ATTRIBUTE_FOUND	No attributes for the index class were found.
937	SBVI_GET_ATTRIBUTE_FAILED	The attribute information could not be obtained.

Table 3 (Page 3 of 3). Error Codes from the VisualInfo for AS/400 High-Level Programming Interface

Error Code	**Message Text**	**Explanation**
938	SBVI_EMPTY_NOTELOG	The note log does not exist.
939	SBVI_GET_TOC_FAILED	The table of contents of the item could not be obtained.
940	SBVI_LOAD_WS_FAILED	The working set could not be loaded.
941	SBVI_CREATE_THREAD_FAILED	A thread could not be created.
942	SBVI_CREATE_DSP_FAILED	An IS/2 window could not be created.
943	SBVI_CREATE_SEM_FAILED	A semaphore could not be created.

Appendix F. Predefined Content Classes

Table 4 lists the predefined content classes for VisualInfo for AS/400.

Table 4 (Page 1 of 3). 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

Table 4 (Page 2 of 3). Predefined Content Classes

Content Class	**Description**
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
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

Table 4 (Page 3 of 3). Predefined Content Classes

Content Class	**Description**
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
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

Bibliography

The following is a bibliography of related IBM publications that you might find helpful while using this book. See “Where to Find More Information” on page vii for information about the VisualInfo for AS/400 product library.

You can request copies of IBM publications from your IBM representative or the IBM branch office serving your area.

IBM AS/400 Publications

Communications and Connectivity

- *IBM AS/400 Communications: Advanced Peer-to-Peer Configuration Guide*, GG24-4023
- *IBM AS/400 Communications: Advanced Peer-to-Peer Network User’s Guide*, SC41-8188
- *IBM AS/400 Communications: Advanced Program-to-Program Communication Programmer’s Guide*, SC41-8189
- *IBM AS/400 Communications: User’s Guide and Reference*, SC09-1168
- *IBM AS/400 Communications Configuration*, SC41-3401
- *IBM AS/400 Communications Management*, SC41-3406
- *IBM AS/400 Network and Systems Management*, SC41-3409
- *IBM AS/400 Network Planning Guide*, GC41-9861

Languages

- *AS/400 Languages: COBOL/400 User’s Guide*, SC09-1812
- *AS/400 Languages: COBOL/400 Reference*, SC09-1813
- *AS/400 Languages: COBOL/400 Reference Summary*, SX09-1285
- *AS/400 Languages: RPG/400 User’s Guide*, SC09-1816
- *AS/400 Languages: RPG/400 Reference*, SC09-1817
- *ILE RPG/400 Reference Summary*, SX09-1261
- *ILE COBOL/400 Programmer’s Guide*, SC09-1522
- *ILE COBOL/400 Reference*, SC09-1523
- *ILE COBOL/400 Reference Summary*, SX09-1260
- *AS/400 Programming: Control Language Programming*, SC41-3721
- *AS/400 Programming: Control Language Reference*, SC41-3722
- *ILE RPG/400 Programmer’s Guide*, SC09-1525
- *ILE RPG/400 Reference*, SC09-1526
- *ILE RPG/400 Reference Summary*, SX09-1261

Planning, Installation, and Migration

- *IBM AS/400 Local Device Configuration*, SC41-3121
- *IBM AS/400: Physical Planning Guide*, GA41-9571
- *IBM AS/400 Software Installation*, SC41-3120
- *IBM AS/400 System Support: Installation Guide - 9404*, SY31-9066
- *IBM System Support AS/400: Installation Guide - 9406* (shipped with the product)

Programming

- *IBM AS/400 Programming: Backup and Recovery Guide*, SC41-8079
- *IBM AS/400 Programming*, SC41-3721
- *IBM AS/400 Programming: Security Concepts and Planning*, SC41-8083
- *Query/400: User’s Guide*, SC41-3210

System Use

- *IBM AS/400 Getting Started on the AS/400 System*, SC41-3204
- *IBM AS/400 PC Support: Technical Reference*, SC41-8091
- *IBM AS/400 Q & A Database Coordinator’s Guide*, SC41-8086
- *IBM AS/400 System Operation*, SC41-3203
- *IBM AS/400 System Operation for New Users*, SC41-3200
- *IBM AS/400 System Operations: Operator’s Guide*, SC41-8082

- *IBM AS/400 System Startup and Problem Handling*, SC41-3206
- *OS/400 Infoseeker Getting Started*, SC41-3001
- *OS/400 Integrated File System Introduction*, SC41-3711

System Management

- *IBM AS/400 Security–Basic*, SC41-3301
- *IBM AS/400 Security–Reference*, SC41-3302
- *OS/400 Backup and Recovery–Basic*, SC41-3304
- *OS/400 Backup and Recovery–Advanced*, SC41-3305

IBM Client Access

Windows

- *IBM Client Access/400 for Windows: Getting Started*, SC41-3530
- *IBM Client Access/400 for Windows: User's Guide*, SC41-3532

IBM ImagePlus VisualInfo

- *IBM ImagePlus VisualInfo Getting Started*, GC31-9051
- *IBM ImagePlus VisualInfo Planning and Installation Guide*, GC31-7772
- *IBM ImagePlus VisualInfo System Administration Guide*, GC31-7774
- *IBM ImagePlus VisualInfo Application Programming Guide for Windows*, GC31-9055
- *IBM ImagePlus VisualInfo Application Programming Guide, Volume 1*, GC31-9063
- *IBM ImagePlus VisualInfo Application Programming Guide, Volume 2*, GC31-9061

IBM 3995 Compact Optical Library Dataserver

Direct-Attached

- *IBM AS/400 Optical Library Dataserver Support/400: User's Guide and Reference*, SC41-0035
- *OS/400 Optical Support*, SC41-4310

LAN-Attached

- *IBM 130 mm Rewritable Optical Disk Cartridge Requirements*, SA37-0261
- *IBM 3995 All LAN Models Reference*, GA32-0147
- *IBM 3995 AS/400 Optical Library Dataserver: Operator's Guide Models A43, 143, 142, 043, and 042*, GA32-0140
- *IBM 3995 Optical Library Dataserver Products: Introduction and Planning Guide for C-Series Models*, GA32-0350
- *IBM 3995 Optical Library Dataserver Products: Model A23 Guide to Operations*, GA32-0291
- *IBM 3995 Optical Library Dataserver Products: Operator's Guide for C-Series Models*, GA32-0352
- *IBM 3995 Optical Library Dataserver Products: User's Guide for Models 123, 122, 121, 113, 112, 111, 023, 022, 021*, GA32-0141

Both Direct- and LAN-Attached

- *IBM 3995 Optical Library Dataserver Products: Introduction and Planning Guide*, GA32-0121
- *IBM 3995 Optical Library Dataserver Products: Optical Disk Format*, GA32-0224
- *IBM 3995 Optical Library Dataserver Products: Optical Disk Cartridge Requirements 130 mm Write-Once 1024 Bytes/Sector*, GA32-0146
- *IBM 3995 Optical Library Dataserver Products: Reference for C-Series Models*, GA32-0351
- *IBM 3995 Optical Library Dataserver Products: Safety*, GA32-0148

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 VisualInfo for AS/400. The items that can be associated with access lists are the data objects stored by users, *index classes* and subsets, *workbaskets*, and *workflows*.

action list. In Workfolder Application Facility work management, an approved list of the actions, defined by a supervisor, that a user can perform on work packages. An action list defines options, such as printing or displaying work packages, and the function keys that are available for use.

active case. A case that was entered and indexed, assigned to a queue, and has been previously pended, unqueued, assigned to a process, or assigned to a workbasket. A case with this status can be accessed through the Work any case, Work queued case, or Review any case menu options.

address ID profile. A control file used in Workfolder Application Facility that contains names and addresses.

ad hoc route. A route that is not part of a defined Workfolder Application Facility work management process. An *ad hoc route* 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.

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 AS/400 system to communicate with programs on other systems having compatible communications support. This communications support is the AS/400 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.

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.

archiving. The storage of backup files and any associated journals, usually for a given period of time.

AS/400. Application System/400.

AS/400 object directory profile. A control file used in Workfolder Application Facility to identify AS/400 object directories used for image document storage.

attribute. Used in VisualInfo for AS/400 APIs, a single value associated with an item (document or folder). Each index class can have up to eight attributes.

automatic importing. The process that operates in the background to complete the importing of documents when it is requested through the facsimile process or the “set up automatic import only” option.

automatic indexing. The indexing process that operates in the background to complete the indexing of documents introduced to the system through the scan and batch index function.

B

binary large object (BLOB). A large stream of binary data treated as a single object.

C

capture. In optical character recognition, to gather picture data from a field on an input document, using a special scan.

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.

case. A uniquely identifiable work item initiated by a user. Cases may be: Active (open and being worked), Pended (suspended awaiting further information), Closed (processing is complete), New, or Not Queued.

case file. A file that contains one entry for each case.

case history file. A file that contains the history of actions against each case.

case ID. A system-assigned identifier that is chronological, based on the time that indexing occurred. Every case has a unique case ID.

client application. An application written with the VisualInfo for AS/400 APIs to customize a user interface.

closed case. A case that was closed and cannot be reopened. A closed case is not assigned to a queue. A case with this status can be accessed through the Review any case menu option.

closing a case. Ending a case permanently. Documents in a closed case may be archived for future access.

collection. (1) In VisualInfo for AS/400, a group of objects with a similar set of storage management rules and contained within a *storage group*. Every object is stored in a collection. (2) In Workfolder Application Facility, it provides categories for entered documents and is used to match incoming documents with outstanding requests. Primarily used in case processing, collections can also be used when indexing documents into file cabinets.

collection point. In Workfolder Application Facility work management, 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 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*.

collection profile. A file that contains one entry for each type of document to be processed.

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.

cross-system importing. The process that imports cases and documents to one Workfolder Application Facility system from another.

cross-system exporting. The process that exports cases and documents from one Workfolder Application Facility system to another.

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.

DASD system ID profile. A file used to define the roles of AS/400 processors in the Workfolder Application Facility system.

DDM. Distributed data management.

DBCS. Double-byte character set.

decision point. In Workfolder Application Facility work management, 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 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*.

destager. A function of the object server that moves objects from the *staging area* to the first step in the object’s *migration policy* or *management class*.

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 AS/400 system.

document. (1) An item containing one or more base parts. See also *collection point*. (2) Information and the medium on which it is recorded that generally have permanence and that humans or machines can read. (3) 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).

document type. Provides categories for entered documents and are used to match incoming documents with outstanding requests. Primarily used in case processing, document types can also be used when indexing documents into file cabinets.

document type profile. A file that contains one entry for each type of document to be processed.

document working set. A set of document images selected from a menu that Workfolder Application Facility provides. This set of document images is sent to the ImagePlus workstation for display.

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

empty case. A case that has no documents.

export. In Workfolder Application Facility, 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 Workfolder Application Facility systems.

F

facsimile configuration profile. A control file that lets the system administrator associate Facsimile Support for OS/400 with Workfolder Application Facility.

fax exporting. A process that operates in the background to send documents requested for fax transmission to a fax server.

fax importing. The importing process that operates in the background to forward fax documents received by the fax server.

FIFO (first in, first out). A queueing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

file cabinet. A component of Workfolder Application Facility that provides document storage and retrieval capabilities designed to help manage selected documents.

file cabinet code. Acts as the file cabinet name.

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.

focus control. In Workfolder Application Facility, a feature that lets the system administrator decide whether Workfolder Application Facility and ImagePlus Workstation Program or the user controls the active window.

folder. (1) In VisualInfo for AS/400, an object that can contain other folders or documents. (2) In Workfolder Application Facility, the area in the AS/400 system where images are stored after successful scanning or importing.

folder balancing. In the AS/400, the process by which documents are distributed evenly among the available folders in the system.

folder manager. In VisualInfo systems other than VisualInfo for AS/400, the term used to describe the data model and a subset of the APIs. In VisualInfo for AS/400, this term refers to the entire set of VisualInfo for AS/400 APIs.

folder name. A 1- to 12-character user-defined word that names a folder. One period (.) is allowed. If the folder name is more than 8 characters, the ninth character must be a period. This can be followed by a 1- to 3-character extension.

folder path. A folder name, followed by one or more additional folder names, where each preceding folder is found. Each folder in the path must be separated by a slash (/). A folder path can consist of 1 to 63 characters.

forms creation utility. A utility that allows the system administrator to capture a preprinted form as an image and to define the content and format of the form.

G

Group III. A compression algorithm that conforms to a standard promulgated by the International Telegraph and Telephone Consultative Committee (CCITT).

H

high-speed indexing. The indexing process that operates in the background to complete the indexing of documents that were input into the system using the high-speed scanning function.

high-speed scanner workstation. A display workstation equipped with a high-speed scanner.

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 spool file. A file that contains sorted, merged, and completed print records ready for print.

image workstation. A programmable workstation that can perform image functions.

import data file. An AS/400 database file that contains data for one or more documents. Using data in the file, Workfolder Application Facility creates documents that can be stored and indexed just like scanned documents.

importing. A process by which documents are input into AS/400 using files rather than the scanning process. Imported documents can be stored in Workfolder Application Facility on DASD and optical, and displayed and printed, in the same manner as scanned documents.

import page ID profile. A file that contains the form overlay and fields specified for each page ID defined for a document type whose data type is 01.

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 with a case or identifier.

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 VisualInfo for AS/400, 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.

indexing. The three-step process consisting of viewing a document, specifying an identifier for the document, and before creating a new case, or matching the document with an existing case.

instance. In work management, an occurrence of a work package within a process. If the process consists of parallel routes, multiple instances of a work package exist.

IOCA. Image Object Content Architecture.

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 workflow. Referred to as an *object* outside of library server functions.

J

journal. A special-purpose file or data set that can be used to provide a record of operator and system actions used to recover data and to identify operator actions that resulted in a problem.

journaling. (1) The process of recording changes made in a physical file member in a journal. Journaling allows you to reconstruct a physical member by applying the changes in the journal to a saved version of the physical file member. (2) The process of recording information sequentially in a database.

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.

keyword field. A field enabled for input that provides data for a single keyword that is defined for a file cabinet.

keyword value. The input specified in the field for each keyword.

L

LAN. Local area network.

language profile. A control file used in Workfolder Application Facility to define country-specific parameters, 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 VisualInfo for AS/400 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 AS/400 application.

M

Machine-Generated Data Structure (MGDS). (1) An IBM structured data format protocol for passing OCR (Optical Character Recognition) data among various applications. When workstations use OCR facility to create coded data from scanned images, those coded data are formatted into MGDS and passed to other applications for further processing. (2) 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.

masking. The action of obscuring part of the image of a document so that it is not visible to the viewer.

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 ImagePlus 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 country. 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 VisualInfo for AS/400 server. Each server must have a network table file that identifies it. The name of the network table is always FRNOLNT.TBL.

new case. A case that was entered and indexed, assigned to a queue, and has not been previously pended, unqueued, assigned to a process, or assigned to a workbasket. A case with this status can be accessed through the Work any case, Work queued case, or Review any case menu options.

NLS. National language support.

not queued case. A case that was entered and indexed, and is not assigned to a queue. A case with this status can be accessed through the Work any case or Review any case menu options.

O

object. (1) An item upon which actions are performed. (2) A collection of data referred to by a single name. (3) The smallest unit within the system. For ImagePlus systems, this is typically a single-image document. (4) Any binary data entity stored on an object server. In the VisualInfo for AS/400 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 Workfolder Application Facility to identify AS/400 object directories used for image document storage.

object server. The component of IBM ImagePlus VisualInfo for AS/400 that physically stores the objects or information that client applications store and access.

OCR. Optical character recognition.

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 character recognition (OCR). Character recognition that uses optical means to identify graphic characters.

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 drive profile. A control file used in Workfolder Application Facility to define the optical controller used for the optical storage of documents.

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 VisualInfo for AS/400 and Workfolder Application Facility. 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*.

output class. A unique name assigned to a specific time frame when faxes are eligible to be transmitted.

output profile. A file that defines the content of each output form.

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.

pend. To suspend a case while awaiting additional information or action, such as a particular document type or date.

pended case. A case that was pended through casework, waiting for more information. A pended case is not assigned to a queue. A case with this status can be accessed through the Work any case or Review any case menu options.

pending. Awaiting further information or action on a case.

platter. See *optical disk.*

prefix. (1) A code dialed by a caller before being connected. (2) A code at the beginning of a message or record.

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. A rank assigned to a task that determines its precedence in receiving system resources.

privilege. An authorization for a user to either access or perform certain tasks on objects stored in VisualInfo for AS/400. The system administrator assigns privileges.

privilege set. (1) In VisualInfo for AS/400, collection of *privileges* for working with system components and functions. The system administrator assigns privilege sets to users (user IDs) and user groups. (2) In the work management system in Workfolder Application Facility, an approved list of the actions, defined by a supervisor, that a user can perform on work packages. An privilege set defines options, such as printing or displaying work packages, and the function keys that are available for use.

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.

Q

queue. A line or list of items waiting to be processed; for example, cases to be worked or messages to be displayed.

queue ID profile. A file that contains one entry for each active case. Each case is indexed by queue ID, queue type, and creation date and time.

R

reindexing. The process of indexing documents that were previously indexed incorrectly. This process is the same as the indexing process.

render. To take data that is not typically image-oriented and depict or display it as an image. In VisualInfo for AS/400, 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. In work management, a set of steps that move work between workbaskets, collection points, and decision points.

S

SBCS. Single-byte character set.

scan overlap. The process by which a document is scanned while a previously scanned document is stored on DASD.

scanner. A device that examines a spatial pattern one part after another and generates analog or digital signals corresponding to the pattern. (I)

scanner workstation. A display workstation equipped with a scanner.

scanning. A physical process that enters documents into an ImagePlus workstation. After a document has been scanned, it can be stored permanently.

scanning and batch indexing. An efficient scanning and indexing option that overlaps the scanning of a document with the storing of another document, while an indexing process operates automatically in the background.

search criteria. In VisualInfo for AS/400, 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. Workfolder Application Facility can convert a spool file to an import data file.

spool writer. The part of the System Support Program that prints output saved in the spool file.

staging. The process of moving a stored object from an offline or low-priority device back to an online or higher priority device, usually on demand of the system or on request of a user. When a user requests an object stored in permanent storage, a working copy is written to the *staging area*.

stand-alone. Pertaining to an operation that is independent of any other device, program, or system.

step number. In work management, the numbers that specify the order in which route commands are processed. Each step number must have an associated command telling the route what action to take.

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. In Workfolder Application Facility, a means of grouping documents together for storage to an optical disk. Workfolder Application Facility provides the following storage methods: file cabinet, collection, prioritized, and system assigned (optical distribution).

storage system. A generic term for storage in VisualInfo for AS/400.

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 a case. To end case processing temporarily.

system administrator. The person who manages the ImagePlus workstation, the Optical Library Subsystem, and the departmental processor. The system administrator helps with problem determination and resolution. Synonymous with *administrator.*

system ID profile. A control file used in Workfolder Application Facility to define the roles of AS/400 processors in the system.

system-managed storage (SMS). The VisualInfo for AS/400 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. (1) Anyone requiring the services of VisualInfo for AS/400. This term generally refers to users of client applications rather than the developers of applications, who use the VisualInfo for AS/400 APIs. (2) In Workfolder Application Facility, the individual who performs input and case processing.

user activity file. A file that contains one entry for each user. It contains item counts and productivity statistics.

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 fields. Data fields defined within the user exit programs. The user exit programs process the data and values passed into these fields.

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. In work management in Workfolder Application Facility for AS/400, a container that holds work packages. Workbaskets can be used as parts of process definitions or ad-hoc routes. In VisualInfo for AS/400, a logical location within the VisualInfo for AS/400 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.

workbasket privilege set. A list that specifies which options and function keys operate on indexing and work with work package panels.

work management. A system that lets an enterprise define a work process and environment to automate workflow and control business processes.

work management case. A case that was entered and indexed, and has been assigned to either a process or a workbasket. A work management case is not assigned to a queue. A case with this status can be accessed through the Review any case, Work with workbaskets, or Search for work packages menu options.

work order. The sequence of work packages in a workbasket.

work package. The work that is routed from one location to another. A work package can consist of an unindexed document, a file cabinet document, a Workfolder Application Facility case, or a user-defined collection of objects. It can also be empty, such as when you first create it and before it contains any work items. 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.

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 predefined type of work package must progress.

workflow. A sequence of *workbaskets* that a document or folder travels through while it is being processed.

working set. A set of pages residing in the workstation, which can constitute one or more documents.

workstation. A computer processor unit, image display unit, scanners, and printers with which the user performs input, indexing, and printing.

Index

Please Tell Us What You Think!

IBM ImagePlus
VisualInfo for AS/400
Application Programming
Guide and Reference
Version 4 Release 1

Publication No. SC34-4586-00

We hope you found this book useful and informative. If you like what we've done, please let us know; if not, please tell us why. We'll use your comments to make the book better.

Please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number to receive a reply.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without obligation.

- To send comments by mail or fax, use the form titled “What Do You Think?” on the following page.

  If you're mailing from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid handling.

  To fax the form, use this number: (919) 254-0206.

- To send comments electronically, use one of the following network IDs:

IBM Mail Exchange	**USIB5DNQ at IBMMAIL**
Internet	**KFRYE@CARVM3.VNET.IBM.COM**

Thank you! Your comments help us make the information more useful for you.

What Do You Think?

IBM ImagePlus
VisualInfo for AS/400
Application Programming
Guide and Reference
Version 4 Release 1

Publication No. SC34-4586-00

We're in business to satisfy you. If we're succeeding, please tell us; if not, let us know how we can do better.

Overall, how satisfied are you with this book?

	Very Satis- fied	Satis- fied	Neither Satis- fied nor Dissatis- fied	Dissat- isfied	Very Dissat- isfied	No Opinion

How satisfied are you that the information in this book is:

Accurate						
Complete						
Easy to find						
Easy to understand						
Well organized						
Applicable to your tasks						

Name

Address

Company or Organization

Phone No.

What Do You Think?
SC34-4586-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Attn: Information Development
Department T71B Building 062
PO Box 12195
Research Triangle Park, NC 27709-2195

Fold and Tape | **Please do not staple** | Fold and Tape

SC34-4586-00

Cut or Fold
Along Line

Program Number: 5733-A18

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-4586-00