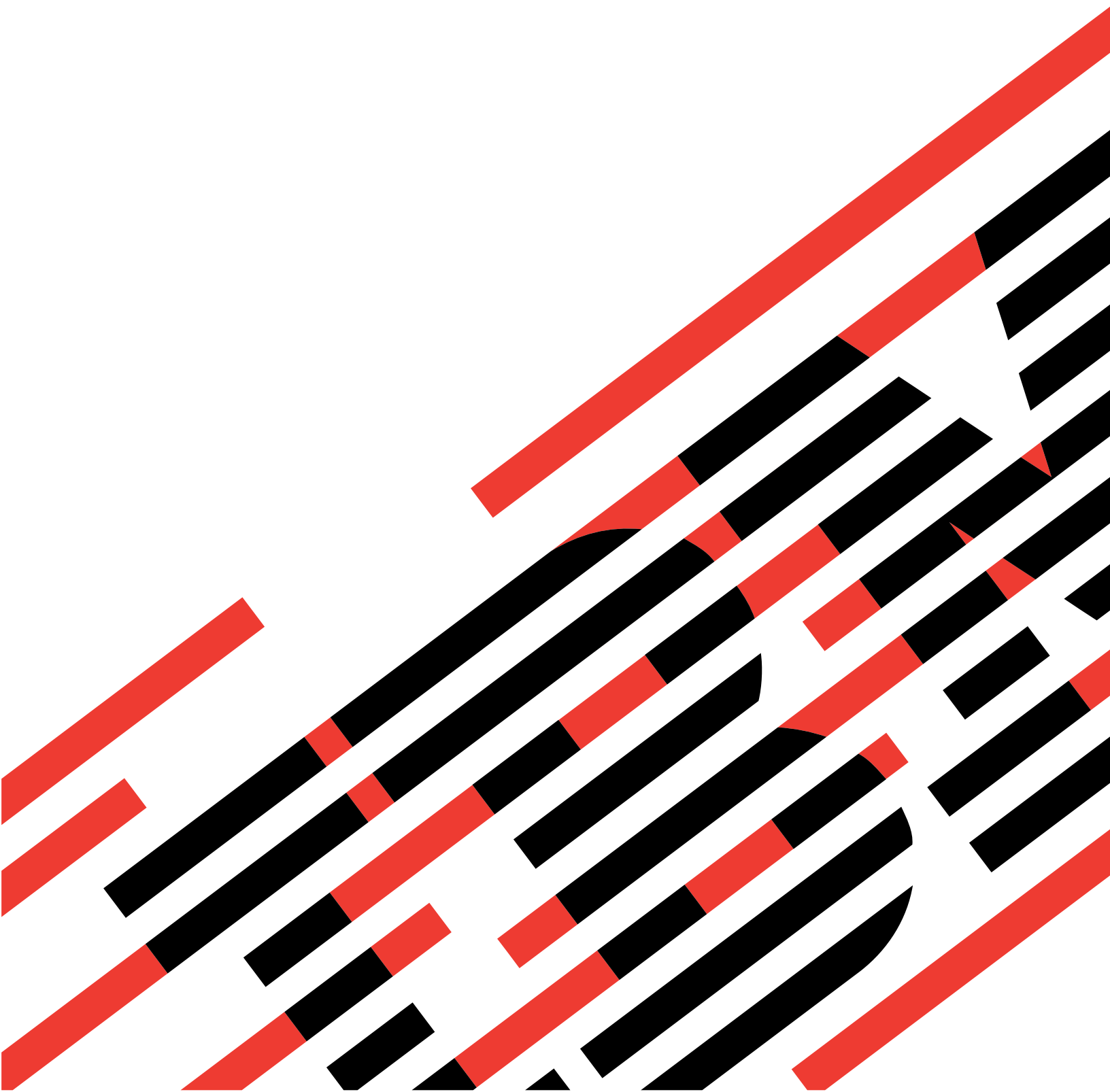




@server

iSeries

iSeries Access for Windows Programming





@server

iSeries

iSeries Access for Windows Programming

Contents

Part 1. iSeries Access for Windows programming	1
Chapter 1. Code disclaimer information	3
Chapter 2. What's new for V5R2	5
Chapter 3. Print this topic	7
Chapter 4. iSeries Access for Windows® C/C++ APIs	9
iSeries Access for Windows C/C++ APIs overview	9
API groups, header files, import libraries, and DLLs	9
iSeries system name formats for ODBC Connection APIs	12
OEM, ANSI, and Unicode considerations	12
Obsolete iSeries Access for Windows APIs	14
Return codes and error messages	16
iSeries Access for Windows Administration APIs	31
Administration APIs listing	32
Example: Administration APIs	40
iSeries Access for Windows Communications and Security APIs	45
System object attributes	45
iSeries Access for Windows Communications and Security system object APIs listing	49
iSeries Access for Windows Communications system list APIs listing	109
Example: Using iSeries Access for Windows communications APIs	132
iSeries Access for Windows Data Queues APIs	144
Data queues	144
Ordering data queue messages	145
Working with data queues	145
Typical use of data queues	145
iSeries Access for Windows Data Queues APIs listing	146
Example: Using Data Queues APIs	206
iSeries Access for Windows Data Transformation and National LanguageSupport (NLS) APIs	207
iSeries Access for Windows data transformation APIs	207
iSeries Access for Windows national language support (NLS) APIs	228
iSeries Access for Windows Directory Update APIs	260
Typical use of iSeries Access for Windows Directory Update APIs	261
Requirements for Directory Update entries	262
Options for Directory Update entries	262
Directory Update package files syntax and format	263
Directory Update sample program	264
iSeries Access for Windows Directory Update API listing	264
iSeries Access for Windows PC5250 emulation APIs	284
IBM Lightweight Directory Access Protocol (LDAP) APIs	284
iSeries Access for Windows Multimedia APIs	285
Ultimedia System Facilities API capabilities overview	285
Ultimedia System Facilities API types overview	286
iSeries Objects APIs for iSeries Access for Windows	286
iSeries objects attributes	287
iSeries Objects API for iSeries Access for Windows listing	316
Example: Using iSeries Objects APIs for iSeries Access for Windows	397
iSeries Access for Windows Remote Command/Distributed Program Call APIs	399
Typical use of iSeries Access for Windows Remote Command/Distributed Program Call APIs	400
iSeries Access for Windows Remote Command/Distributed Program Call APIs listing	401
Example: Using Remote iSeries Access for Windows Command/Distributed Program Call APIs	420

iSeries Access for Windows Serviceability APIs	422
History log and trace files	423
Error handles	424
Typical use of Serviceability APIs	424
iSeries Access for Windows Serviceability APIs listing	424
Example: Using iSeries Access for Windows erviceability APIs	490
iSeries Access for Windows System Object Access (SOA) APIs	491
SOA objects	492
iSeries object views	492
Typical use of System Object Access APIs for iSeries Access for Windows	492
iSeries Access for Windows System Object Access programming considerations.	500
System Object Access APIs for iSeries Access for Windows listing	501
Chapter 5. iSeries Access for Windows Database Programming	555
iSeries Access for Windows OLE DB Provider	555
iSeries Access for Windows ODBC	556
ODBC APIs	557
Implementation issues of ODBC APIs	586
iSeries Access for Windows ODBC performance	602
Choosing an interface to access the ODBC driver	639
ODBC programming examples	640
iSeries Access for Windows database APIs	646
iSeries Access for Windows database APIs overview	647
Typical use of iSeries Access for Windows database APIs	649
Objects that process data on the PC or iSeries server	650
Code page support in Windows	651
iSeries Access for Windows database APIs listing	652
Example: Using SQL to access database functions	845
Chapter 6. Java programming	849
Chapter 7. ActiveX programming	851

Part 1. iSeries Access for Windows programming

As an iSeries™ application developer, explore this topic to reference and use iSeries Access for Windows technical programming information, tools, and techniques. This information includes programming concepts, capabilities, and examples that are useful when writing applications to access the resources of an iSeries server. If a basic working knowledge of iSeries Access for Windows and its components is needed see the **Welcome Wizard** and the **User's Guide**, which are shipped with iSeries Access for Windows.

Note: To launch these components from a Windows PC, select **Start** —> **Programs** —> **IBM iSeries Access for Windows**, and select the component. If you do not see either of the components in your iSeries Access for Windows folder, they are not installed. Run **Selective Setup** to install them.

See the iSeries Access for Windows - Setup  book for related information. (Welcome Wizard is always installed).

Client/server applications can be developed and tailored to the needs of your business using this topic. Various programming techniques are described so you can connect, manage, and take advantage of the rich functions provided by the server.

Find information for your iSeries Access for Windows programming needs by selecting from the following topics:

What's new for V5R2

Find a summary of the new function that is included in the programming topics for this release.

Print this topic

Find how to view and print a PDF version of iSeries Access for Windows programming.

C/C++ application programming interfaces

Find APIs to access the iSeries server from your client-based applications.

Database programming (OLE DB Provider, ODBC and Database APIs)

Find tips and techniques on database interfaces. You can access iSeries database files and stored procedures and use them to perform various server tasks. Although C/C++ interfaces are shown, knowing C/C++ is not a requirement for the use of the APIs associated with these interfaces.

Java™ programming

Find information on developing web-based applications using Java programming.

ActiveX programming

Find how to use ActiveX programming methods to access iSeries resources through the use of ActiveX automation technology.

Plug-ins for iSeries Navigator

Find a convenient way to integrate your own functions and applications into a single user interface.

Programmer's Toolkit

Find a primary information source for developing applications with iSeries Access for Windows.

Chapter 1. Code disclaimer information

This document contains programming examples.

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Chapter 2. What's new for V5R2

Chapter 5, "iSeries Access for Windows Database Programming" on page 555

This page highlights changes to iSeries Access for Windows programming topic for V5R2.

New database programming support

For V5R2, both OLE DB and ODBC support 64-bit functions as well as 32-bit functions. Database APIs have been updated to support several new functions and two new ODBC drivers.



- 64-bit support for all 32-bit functions (For restrictions see Restrictions when using the 64-bit iSeries Access for Windows ODBC Driver.)
- iSeries Access for Windows C/C++ APIs now offer support for the following:
 - Independent disk pools (also known as independent ASPs)
 - Row identifier information (ROWID)
 - Extended column information
 - Kerberos principal
- There are now two new supported drivers for iSeries Access for Windows ODBC drivers.
 - 64-bit ODBC driver
 - Linux ODBC driver


Changed MAPI support

V5R2M0 iSeries Access for Windows does not support Messaging Application Programming Interface (MAPI). If you were using MAPI for the purpose of having your mail application access the system distribution directory (SDD), it is recommended that you now use Lightweight Directory Access Protocol (LDAP) instead. iSeries Navigator provides configuration support so that SDD information is published to LDAP.

How to see what's new or changed

To help you see where technical changes have been made, this information uses:


- The  image to mark where new or changed information begins.
- The  image to mark where new or changed information ends.

To find other information about what's new or changed this release, see the Memo to Users. 

Chapter 3. Print this topic

You can view, print, and download a PDF version of iSeries Access for Windows programming. You must have Adobe® Acrobat® Reader installed to view PDF files. Download a copy of Acrobat from

<http://www.adobe.com/prodindex/acrobat/readstep.html>  .

To view, print, and download the PDF version, select iSeries Access for Windows programming  (about 2.25 MB, or 848 pages). Print the entire document, or select and print a range of pages.

To save a PDF on your workstation for viewing or printing:

1. Open the PDF in your browser (select the link above).
2. In the menu of your browser, select **File**.
3. Select **Save As...**
4. Navigate to the directory in which you would like to save the PDF.
5. Select **Save**.

Chapter 4. iSeries Access for Windows® C/C++ APIs

iSeries Access for Windows provides C/C++ application programming interfaces (APIs) for accessing iSeries resources. These APIs are intended primarily for C/C++ programmers. However, they also may be called from other languages that support calling C-style APIs.

iSeries Access for Windows C/C++ APIs overview information:

“iSeries Access for Windows C/C++ APIs overview”

iSeries Access for Windows C/C++ APIs topics:

- Administration — Listing of APIs
- Communications and Security — Listing of APIs
- Database programming (OLE DB Provider, ODBC and Database APIs)
 - Listing of ODBC APIs
- iSeries Data Queues — Listing of APIs
- Data transformation and national language support(NLS) — Listing of APIs
- Directory Update
- PC5250 emulation
- IBM® Lightweight Directory Access Protocol (LDAP)
- Multimedia
- iSeries Objects
- Remote Command/Distributed Program Call
- Serviceability
- System Object Access (SOA)

Note: Read the Chapter 1, “Code disclaimer information” on page 3 for important legal information.

iSeries Access for Windows C/C++ APIs overview

See the following topics for iSeries Access for Windows C/C++ APIs overview information:

- “API groups, header files, import libraries, and DLLs”
- “iSeries system name formats for ODBC Connection APIs” on page 12
- “OEM, ANSI, and Unicode considerations” on page 12
- “Obsolete iSeries Access for Windows APIs” on page 14
- “Return codes and error messages” on page 16

API groups, header files, import libraries, and DLLs

For each iSeries Access for Windows C/C++ API group, the table below provides:

- Links to the API documentation
- Required interface definition (header) files, where applicable
- Associated import library files, where applicable
- Associated Dynamic Link Library (DLL) files

Access interface definition files for all iSeries Access for Windows C/C++ API groups in the iSeries Access for Windows **Programmer’s Toolkit**.

How to access iSeries Access for Windows header files in the Toolkit:

1. Find the **Programmer’s Toolkit** icon in your iSeries Access for Windows program directory and launch it. If it is not displayed in the program directory, install the Toolkit.
2. In the left navigation panel, select the appropriate API group.

Note: Names of some API categories in the Programmer’s Toolkit differ from the names that are used in iSeries Access for Windows programming:

To find this iSeries Access for Windows programming API group header file:	Select this Programmer's Toolkit topic:
Administration	Client Information
Data transformation	Data Manipulation
National language support	
LDAP	Directory
Serviceability	Error Handling
AS/400® Object	AS/400 Operations
System Object Access	

3. Select the **C/C++ APIs** subtopic in the left navigation panel.
4. In the right display panel, find the header (.h) file and select it.

Note: In addition to interface descriptions and definitions, the iSeries Access for Windows API group topics in the Toolkit include links to other information resources.

About import libraries:

The import libraries that are shipped with the Programmer's Toolkit were built with the Microsoft® Visual C++ compiler. As a result, they are in the Common Object File Format (COFF). Some compilers, such as Borland's C compiler, do not support COFF. To access the iSeries Access for Windows C/C++ APIs from these compilers, you must create Object Model Format (OMF) import libraries by using the IMPLIB tool. For example:


```
implib cwbdc.lib %windir%\system32\cwbdc.dll
```

Note: As of V5R1, the format for certain import libraries has changed to make the file sizes smaller. This includes cwbapi.lib and fzzmapiwlib. These libraries will not work with Microsoft Visual C++ 5.0 or earlier. If you need to call the APIs from Microsoft Visual C++ 5.0 or earlier, you can get the import libraries built using the old format at import libraries (<http://www.ibm.com/eserver/iseries/access/toolkit/importlibraries.htm>)

Table 1. iSeries Access for Windows C/C++ API groups, header files, library files, and DLL files

API group	Header file	Import library	DLL
Administration	cwbad.h	cwbapi.lib	cwbad.dll
Communications and Security	cwbcosys.h cwbcos.h cwb.h	cwbapi.lib	cwbcos.dll
AS/400 Data Queues	cwbdc.h	cwbapi.lib	cwbdc.dll
Data transformation	cwbdt.h	cwbapi.lib	cwbdt.dll
Directory Update	cwbup.h	cwbapi.lib	cwbup.dll
Emulation (Standard HLLAPI interface)	hapi_c.h	pascal32.lib	pcshll.dll pcshll32.dll
Emulation (Enhanced HLLAPI interface)	ehlapi32.h	ehlapi32.lib	ehlapi32.dll

Table 1. iSeries Access for Windows C/C++ API groups, header files, library files, and DLL files (continued)


API group	Header file	Import library	DLL
Emulation (Windows EHLLAPI interface)	whllapi.h	whllapi.lib	whllapi.dll
		whlapi32.lib	whllapi32.dll
Emulation (HACL interface)	eclall.hpp	pcseclva.lib	pcseclva.dll
		pcseclvc.lib	pcseclvc.dll
Emulation (PCSAPI interface)	pcsapi.h	pcscal32.lib	pcsapi.dll pcsapi32.dll
Multimedia	fzzmapi.h	fzzmapiw.lib	fzzmapiw.dll
National language support (General NLS)	cwbnl.h	cwbapi.lib	cwbnl.dll
National language support (Conversion NLS)	cwbnlcnv.h	cwbapi.lib	cwbnl1.dll
National language support (Dialog-box NLS)	cwbnldlg.h	cwbapi.lib	cwbnldlg.dll
AS/400 objects	cwbobj.h	cwbapi.lib	cwbobj.dll
ODBC	sql.h sqlext.h sqltypes.h sqlucode.h	odbc32.lib	odbc32.dll
Database APIs (Optimized SQL)	cwbdb.h	cwbapi.lib	cwbdb.dll
OLE DB Provider	ad400.h da400.h		cwbzzodb.dll See the OLE DB Section of the Microsoft Universal Data Access Web Site  for more information
Remote Command/Distributed Program Call	cwbrc.h	cwbapi.lib	cwbrc.dll
Serviceability	cwbsv.h	cwbapi.lib	cwbsv.dll
System Object Access	cwboapi.h	cwbapi.lib	cwboapi.dll

Who should read iSeries Access for Windows programming

This information is designed for client/server application developers who have a basic working knowledge of iSeries Access for Windows and its components. For detailed information about iSeries Access for Windows and its components, see the **Welcome Wizard** and the **User's Guide**, which are shipped with iSeries Access for Windows.

Note: To launch these components from a Windows PC, select **Start** → **Programs** → **IBM iSeries Access for Windows**, and select the component. If you do not see either of the components in

your iSeries Access for Windows folder, they are not installed. Run **Selective Setup** to install them.

See the iSeries Access for Windows - Setup  book for related information.

Programmer's Toolkit

The iSeries Access for Windows Programmer's Toolkit—an installable component of iSeries Access for Windows—should be used as the primary source of information about iSeries Access for Windows application development. This includes programming with iSeries Access for Windows ActiveX Automation Objects, ADO/OLE DB, and Java. The Programmer's Toolkit contains links to header files, sample programs, and complete documentation.

Note: No portion of the Toolkit or the iSeries Access for Windows product may be redistributed with the resulting applications.

The Programmer's Toolkit consists of two parts:

Programmer's Toolkit component of iSeries Access for Windows

This includes:

- The Toolkit help file and other Windows help documentation
- C/C++ header files
- C import libraries
- ActiveX automation type libraries
- iSeries ADO Wizards for Visual Basic for the iSeries Access for Windows OLE DB provider

Programmer's Toolkit Web site

This includes sample applications and tools that may be useful for developing iSeries Access for Windows applications. This site is updated regularly; check it periodically for new information.

Follow these links for instructions on how to install and launch the Programmer's Toolkit.

Installing the Programmer's Toolkit: To install the Programmer's Toolkit:

1. If you are installing iSeries Access for Windows for the first time, perform an iSeries Access for Windows Custom Install. If iSeries Access for Windows already is installed, select **Start** → **Programs** → **IBM iSeries Access for Windows** → **Selective Setup**.
2. Follow the prompts until the **Component Selection** dialog displays.
3. Select the **Programmer's Toolkit** option, and follow the prompts to completion.

See also "Launching the Programmer's Toolkit".

Launching the Programmer's Toolkit: To launch the Programmer's Toolkit, select **Start** → **Programs** → **IBM iSeries Access for Windows** → **Programmer's Toolkit**.

Note: The iSeries Access for Windows installation program does not create the Toolkit icon unless you have installed the Programmer's Toolkit on your personal computer. See "Installing the Programmer's Toolkit" for instructions.

iSeries system name formats for ODBC Connection APIs

APIs that take an iSeries system name as a parameter accept names in the following formats:

- TCP/IP network name (system.network.com)
- System name without a network identifier (SYSTEM)
- IP address (1.2.3.4)

OEM, ANSI, and Unicode considerations

Most of the iSeries Access for Windows C/C++ APIs that accept string parameters exist in three forms:

- One that expects string parameters to be expressed in the OEM code page (the default)

- One that expects string parameters to be expressed in the ANSI code page
- One that expects string parameters to be expressed in Unicode

The generic version of the iSeries Access for Windows C/C++ APIs follows the same form as the default OEM version. Only a single name for each function appears in this information, but there are three different system entry points. For example:

```
cwbNL_GetLang();
```

compiles to:

```
cwbNL_GetLang(); //CWB_OEM or undefined
```

or:

```
cwbNL_GetLangA(); //CWB_ANSI defined
```

or:

```
cwbNL_GetLangW(); //CWB_UNICODE defined
```

API types, name formats, and pre-processor definitions

API type	API name format (if it exists)	Pre-processor definition
OEM	cwbXX_xxx	None (may specify CWB_OEM explicitly)
ANSI	cwbXX_xxxA	CWB_ANSI
UNICODE	cwbXX_xxxW	CWB_UNICODE

Note:

- Data transformation APIs (**cwbDT_xxx**) do not follow the "A" and "W" suffix conventions. The generic version of the APIs uses "String" as part of the function name. The ANSI/OEM version uses "ASCII" as part of the function name. The Unicode version uses "Wide" as part of the function name. There is no difference between OEM and ANSI character sets in **cwbDT_xxx** APIs, which handle numeric strings. Therefore, ANSI and OEM versions of the relevant APIs are the same. For example:

```
cwbDT_HexToString();
```

compiles to:

```
cwbDT_HexToASCII(); //CWB_UNICODE not defined
```

or:

```
cwbDT_HexToWide(); //CWB_UNICODE defined
```

See the data transformation **cwbdt.h** header file for more details.

- For Unicode APIs that take a buffer and a length for passing strings (for example, **cwbCO_GetUserIDExW**), the length is treated as the number of bytes. It is not treated as the number of characters.

Using single and mixed API types:

You can write applications that use a single API type, or that combine several API types. Link to the following topics for more information:

- "Using a single iSeries Access for Windows API type" on page 14
- "Using mixed iSeries Access for Windows API types" on page 14

Writing generic applications:

To ensure maximum portability of your applications, consider writing a generic application. Link to the following topic for more information:

- “Writing a generic iSeries Access for Windows application”

Using a single iSeries Access for Windows API type

To restrict your application to a particular type of iSeries Access for Windows API, you must define one—and only one—of the following pre-processor definitions:

```
CWB_OEM_ONLY
CWB_ANSI_ONLY
CWB_UNICODE_ONLY
```

For example, when writing a pure ANSI application, you specify both `CWB_ANSI_ONLY` and `CWB_ANSI`. Refer to the individual Programmer’s Toolkit header files for details of these pre-processor definitions and API names. See “API groups, header files, import libraries, and DLLs” on page 9 for more information.

Using mixed iSeries Access for Windows API types

You can mix ANSI, OEM, and Unicode APIs by using explicit API names. For example, you can write an ANSI iSeries Access for Windows application by specifying the `CWB_ANSI` pre-processor definition, but still call a Unicode version of an API by using the “W” suffix.



Writing a generic iSeries Access for Windows application

Generic applications allow maximum portability because the same source code can be compiled for OEM, ANSI, and Unicode. Generic applications are built by specifying different pre-processor definitions, and by using the generic version of the iSeries access for Windows APIs (the ones without the “A” or “W” suffixes). Following is a short list of guidelines for writing a generic application:

- Instead of including the usual `<string.h>` for manipulating strings, include `<TCHAR.H>`.
- Use generic data types for characters and strings. Use `'TCHAR'` for `'char'` in your source code.
- Use the `_TEXT` macro for literal characters and strings. For example, `TCHAR A[] = _TEXT("A Generic String")`.
- Use generic string manipulation functions. For example, use `_tcscpy` instead of `strcpy`.
- Be especially careful when using the `'sizeof'` operator - always remember that a Unicode character occupies two bytes. When determining the number of characters in a generic `TCHAR` array `A`, instead of the simple `sizeof(A)`, use `sizeof(A)/sizeof(TCHAR)`.
- Use proper pre-processor definitions for compilation. When compiling your source for Unicode in Visual C++, you should also use the pre-processor definitions `UNICODE` and `_UNICODE`. Instead of defining `_UNICODE` in the MAK file, you may want to define it at the beginning of your source code as:

```
#ifndef UNICODE
#define _UNICODE
#endif
```

For a complete description of these guidelines, see the following resources:

1. Richter, J. *Advanced Windows: The Developer’s Guide to the Win32 API for Windows NT® 3.5 and Windows 95*, Microsoft Press, Redmond, WA, 1995.
2. Kano, Nadine *Developing International Software for Windows 95 and Windows NT: a handbook for software design*, Microsoft Press, Redmond, WA, 1995.
3. Microsoft Knowledge Base  articles
4. MSDN Library 

Obsolete iSeries Access for Windows APIs

- | Some of the APIs that were provided by Client Access have been replaced with new APIs in this release.
- | While these older, obsolete APIs are still supported, it is recommended that you use the newer iSeries Access for Windows APIs.

Following is a list, by function, of obsolete APIs. For each obsolete API, a link to the newer iSeries Access for Windows replacement API is provided, when available.

Note: All of the **APPC** and **License Management** APIs are obsolete, and are not supported for iSeries Access for Windows.

Obsolete iSeries Access APIs listing:

- “Obsolete Communications APIs”
- “Obsolete Data Queues APIs”
- “Obsolete Remote Command/Distributed Program Call APIs”
- “Obsolete Security APIs”
- “Obsolete System Object Access (SOA) API” on page 16

Obsolete Communications APIs

- `cwbCO_IsSystemConfigured` (not available)

iSeries Access for Windows does not require pre-configuration of an iSeries server connection to connect to and use that system. For this reason, programs that need to connect to an iSeries server (either explicitly, by calling `cwbCO_Connect`, or implicitly, as the result of a call to a different API such as `cwbRC_RunCmd`) do not need to check to see if the connection has been pre-configured. Therefore, the above API no longer should be necessary.

- **`cwbCO_IsSystemConnected`** (use “`cwbCO_IsConnected`” on page 85)

Most iSeries Access for Windows APIs work with iSeries System Objects, rather than with iSeries server names. There can be multiple iSeries System Objects created and connected to the same iSeries server within the same process. The `cwbCO_IsSystemConnected` API will return an indication of whether at least one System Object is connected to the iSeries server, within the current process. The `cwbCO_IsConnected` API is used to determine if a specific iSeries System Object is connected.

- **`cwbCO_GetUserID`** (use “`cwbCO_GetUserIdx`” on page 81)

Most iSeries Access for Windows APIs work with iSeries System Objects, rather than with iSeries server names. There can be multiple iSeries System Objects created and connected to the same iSeries server, within the same process, but using different user IDs. The `cwbCO_GetUserID` API will return the user ID of the first iSeries System Object, in the current process, for the specified iSeries server. The `cwbCO_GetUserIdx` API will return the user ID for a specific iSeries System Object.

- **`cwbCO_GetHostVersion`** (use “`cwbCO_GetHostVersionEx`” on page 71)

The behavior of these APIs is the same. However, use of the `cwbCO_GetHostVersionEx` API is more efficient.

Obsolete Data Queues APIs

- `cwbDQ_Create` (use “`cwbDQ_CreateEx`” on page 155)
- `cwbDQ_Delete` (use “`cwbDQ_DeleteEx`” on page 161)
- `cwbDQ_Open` (use “`cwbDQ_OpenEx`” on page 189)
- `cwbDQ_StartSystem` (use “`cwbCO_Connect`” on page 59)

Note: To achieve the same effect as `cwbDQ_StartSystem` when you use `cwbCO_Connect`, you must connect to the data queue’s service. See “`cwbCO_Connect`” on page 59 for details.

- `cwbDQ_StopSystem` (use “`cwbCO_Disconnect`” on page 65)

Note: To achieve the same effect as `cwbDQ_StopSystem` when you use `cwbCO_Disconnect`, you must disconnect from the data queue’s service. See “`cwbCO_Disconnect`” on page 65 for details.

Obsolete Remote Command/Distributed Program Call APIs

- `cwbRC_StartSys` (use “`cwbRC_StartSysEx`” on page 418).
- `cwbRC_GetSysName` (use “`cwbCO_GetSystemName`” on page 80).

Obsolete Security APIs

- `cwbSY_CreateSecurityObj` (use “`cwbCO_CreateSystem`” on page 61).
- `cwbSY_DeleteSecurityObj` (use “`cwbCO_DeleteSystem`” on page 64).

- cwbsY_SetSys (use “cwbcO_CreateSystem” on page 61 and pass a system name on the call).
- cwbsY_VerifyUserIDPwd (use “cwbcO_VerifyUserIDPassword” on page 106).
- cwbsY_ChangePwd (use “cwbcO_ChangePassword” on page 57).
- cwbsY_GetUserID (use “cwbcO_GetUserIDEx” on page 81).
- cwbsY_Logon (use “cwbcO_Signon” on page 102).
- cwbsY_LogonUser (use “cwbcO_SetUserIDEx” on page 99, “cwbcO_SetPassword” on page 93, or “cwbcO_Signon” on page 102).
- cwbsY_GetDateTimeCurrentSignon (use “cwbcO_GetSignonDate” on page 79)
- cwbsY_GetDateTimeLastSignon (use “cwbcO_GetPrevSignonDate” on page 77)
- cwbsY_GetDateTimePwdExpires (use “cwbcO_GetPasswordExpireDate” on page 74)
- cwbsY_GetFailedAttempts (use “cwbcO_GetFailedSignons” on page 69)

Obsolete Serviceability APIs

The following Serviceability APIs for reading problem log service records are obsolete:

- cwbsV_GetCreatedBy (not available)
- cwbsV_GetCurrentFix (not available)
- cwbsV_GetFailMethod (not available)
- cwbsV_GetFailModule (not available)
- cwbsV_GetFailPathName (not available)
- cwbsV_GetFailProductID (not available)
- cwbsV_GetFailVersion (not available)
- cwbsV_GetOriginSystemID (not available)
- cwbsV_GetOriginSystemIPAddr (not available)
- cwbsV_GetPreviousFix (not available)
- cwbsV_GetProblemID (not available)
- cwbsV_GetProblemStatus (not available)
- cwbsV_GetProblemText (not available)
- cwbsV_GetProblemType (not available)
- cwbsV_GetSeverity (not available)
- cwbsV_GetSymptomString (not available)

Obsolete System Object Access (SOA) API

CWBSO_CreateListHandle (use “CWBSO_CreateListHandleEx” on page 508)

Return codes and error messages

The iSeries Access for Windows C/C++ application programming interfaces (APIs) support the return of an integer return code on most functions. The return codes indicate how the function completed.

iSeries Access for Windows return codes categories:

- “iSeries Access for Windows return codes that correspond to operating system errors” on page 17
- “iSeries Access return codes” on page 18
- “iSeries Access for Windows component-specific return codes” on page 22

iSeries Access for Windows logs error messages in the History Log, and on the iSeries System.

Error messages in the History Log:

Starting the History Log:

By default, the History Log is not active. To ensure that error messages are written to this file, History logging must be started. See the iSeries Access for Windows User’s Guide, which is shipped with iSeries Access for Windows, for information on starting the History Log

Viewing logged messages:

To view messages that have been logged in the History Log, select **Start —> Programs —>iSeries Access for Windows—> Service —> History Log**.

The entries in the History Log consist of messages with and without message IDs. Messages with message IDs have online help available. Messages without message IDs do not have online help available. To display the cause and recovery information associated with a message that has a message ID, double-click on it. You also can view any message that has a message ID by selecting the Message topic in the online iSeries Access for Windows User's Guide.

Error messages on the iSeries system:

iSeries Access for Windows also has associated messages that are logged on the iSeries server. These messages begin with PWS or IWS. To display a specific PWSxxxx or IWSxxxx message, type the appropriate command at the iSeries command line prompt, where xxxx is the number of the message:

```
DSPMSGD RANGE(IWSxxxx) MSGF(QIWS/QIWSMSG)
```

```
DSPMSGD RANGE(PWSxxxx) MSGF(QIWS/QIWSMSG)
```

iSeries Access for Windows return codes that correspond to operating system errors

0	CWB_OK	Successful completion.
1	CWB_INVALID_FUNCTION	Function not supported.
2	CWB_FILE_NOT_FOUND	File not found.
3	CWB_PATH_NOT_FOUND	Path not found.
4	CWB_TOO_MANY_OPEN_FILES	The system cannot open the file.
5	CWB_ACCESS_DENIED	Access is denied.
6	CWB_INVALID_HANDLE	The list handle is not valid.
8	CWB_NOT_ENOUGH_MEMORY	Insufficient memory, may have failed to allocate a temporary buffer.
15	CWB_INVALID_DRIVE	The system cannot find the drive specified.
18	CWB_NO_MORE_FILES	No more files are found.
21	CWB_DRIVE_NOT_READY	The device is not ready.
31	CWB_GENERAL_FAILURE	General error occurred.
32	CWB_SHARING_VIOLATION	The process cannot access the file because it is being used by another process.
33	CWB_LOCK_VIOLATION	The process cannot access the file because another process has locked a portion of the file.
38	CWB_END_OF_FILE	End of file has been reached.
50	CWB_NOT_SUPPORTED	The network request is not supported.
53	CWB_BAD_NETWORK_PATH	The network path was not found.
54	CWB_NETWORK_BUSY	The network is busy.
55	CWB_DEVICE_NOT_EXIST	The specified network resource or device is no longer available.
59	CWB_UNEXPECTED_NETWORK_ERROR	An unexpected network error occurred.
65	CWB_NETWORK_ACCESS_DENIED	

Network access is denied.
 80 CWB_FILE_EXISTS
 The file exists.
 85 CWB_ALREADY_ASSIGNED
 The local device name is already in use.
 87 CWB_INVALID_PARAMETER
 A parameter is invalid.
 88 CWB_NETWORK_WRITE_FAULT
 A write fault occurred on the network.
 110 CWB_OPEN_FAILED
 The system cannot open the device or file specified.
 111 CWB_BUFFER_OVERFLOW
 Not enough room in the output buffer. Use *bufferSize to determine
 the correct size.
 112 CWB_DISK_FULL
 There is not enough space on the disk.
 115 CWB_PROTECTION_VIOLATION
 Access is denied.
 124 CWB_INVALID_LEVEL
 The system call level is not correct.
 142 CWB_BUSY_DRIVE
 The system cannot perform a JOIN or SUBST at this time.
 252 CWB_INVALID_FSD_NAME
 The device name is incorrect.
 253 CWB_INVALID_PATH
 The network path specified is incorrect.

iSeries Access return codes

The following return codes apply only to iSeries Access:

- “Global iSeries Access return codes”
- “iSeries Access for Windows -specific return codes” on page 19

Global iSeries Access return codes:

4000 CWB_USER_CANCELLED_COMMAND
 Command cancelled by user.
 4001 CWB_CONFIG_ERROR
 A configuration error has occurred.
 4002 CWB_LICENSE_ERROR
 A license error has occurred.
 4003 CWB_PROD_OR_COMP_NOT_SET
 Internal error due to failure to properly register and use a
 product or component.
 4004 CWB_SECURITY_ERROR
 A security error has occurred.
 4005 CWB_GLOBAL_CFG_FAILED
 The global configuration attempt failed.
 4006 CWB_PROD_RETRIEVE_FAILED
 The product retrieve failed.
 4007 CWB_COMP_RETRIEVE_FAILED
 The computer retrieve failed.
 4008 CWB_COMP_CFG_FAILED
 The computer configuration failed.
 4009 CWB_COMP_FIX_LEVEL_UPDATE_FAILED
 The computer fix level update failed.
 4010 CWB_INVALID_API_HANDLE
 Invalid request handle.
 4011 CWB_INVALID_API_PARAMETER
 Invalid parameter specified.
 4012 CWB_HOST_NOT_FOUND
 AS/400 system inactive or does not exist.
 4013 CWB_NOT_COMPATIBLE
 Client Access program or function not at correct level.
 4014 CWB_INVALID_POINTER
 A pointer is NULL.
 4015 CWB_SERVER_PROGRAM_NOT_FOUND
 AS/400 application not found.

4016 CWB_API_ERROR
General API failure.

4017 CWB_CA_NOT_STARTED
Client Access has not been started.

4018 CWB_FILE_IO_ERROR
Record could not be read.

4019 CWB_COMMUNICATIONS_ERROR
A communications error occurred.

4020 CWB_RUNTIME_CONSTRUCTOR_FAILED
The C Run-time constructor failed.

4021 CWB_DIAGNOSTIC
Unexpected error. Record the message number and data in the message and contact IBM Support.

4022 CWB_COMM_VERSION_ERROR
Data queues will not run with this version of communications.

4023 CWB_NO_VIEWER
The viewer support for Client Access/400 was not installed.

4024 CWB_MODULE_NOT_LOADABLE
A filter DLL was not loadable.

4025 CWB_ALREADY_SETUP
Object has already been set up.

4026 CWB_CANNOT_START_PROCESS
Attempt to start process failed. See other error code(s).

4027 CWB_NON_REPRESENTABLE_UNICODE_CHAR
One or more input UNICODE characters have no representation in the code page that is being used.

8998 CWB_UNSUPPORTED_FUNCTION
The function is unsupported.

8999 CWB_INTERNAL_ERROR
An internal error occurred.

iSeries Access for Windows -specific return codes:

- “Security return codes”
- “Communications return codes” on page 20
- “Configuration return codes” on page 20
- “Automation Object return codes” on page 20
- “WINSOCK return codes” on page 20
- “SSL return codes” on page 21

Security return codes:

8001 CWB_UNKNOWN_USERID

8002 CWB_WRONG_PASSWORD

8003 CWB_PASSWORD_EXPIRED

8004 CWB_INVALID_PASSWORD

8006 CWB_INCORRECT_DATA_FORMAT

8007 CWB_GENERAL_SECURITY_ERROR

8011 CWB_USER_PROFILE_DISABLED

8013 CWB_USER_CANCELLED

8014 CWB_INVALID_SYSNAME

8015 CWB_INVALID_USERID

8016 CWB_LIMITED_CAPABILITIES_USERID

8019 CWB_INVALID_TP_ON_HOST

8022 CWB_NOT_LOGGED_ON

8026 CWB_EXIT_PGM_ERROR

8050 CWB_TIMESTAMP_NOT_SET

8257 CWB_PW_TOO_LONG

8258 CWB_PW_TOO_SHORT

8259 CWB_PW_REPEAT_CHARACTER

8260 CWB_PW_ADJACENT_DIGITS

8261 CWB_PW_CONSECUTIVE_CHARS

8262 CWB_PW_PREVIOUSLY_USED

8263 CWB_PW_DISALLOWED_CHAR

8264 CWB_PW_NEED_NUMERIC

8266 CWB_PW_MATCHES_OLD
8267 CWB_PW_NOT_ALLOWED
8268 CWB_PW_CONTAINS_USERID
8270 CWB_PW_LAST_INVALID_PWD

Communications return codes:

8400 CWB_INV_AFTER_SIGNON
8401 CWB_INV_WHEN_CONNECTED
8401 CWB_INV_BEFORE_VALIDATE
8403 CWB_SECURE_SOCKETS_NOTAVAIL
8404 CWB_RESERVED1
8405 CWB_RECEIVE_ERROR
8406 CWB_SERVICE_NAME_ERROR
8407 CWB_GETPORT_ERROR
8408 CWB_SUCCESS_WARNING
8409 CWB_NOT_CONNECTED
8410 CWB_DEFAULT_HOST_CCSD_USED

Configuration return codes:

8500 CWB_RESTRICTED_BY_POLICY
8501 CWB_POLICY_MODIFY_MANDATED_ENV
8502 CWB_POLICY_MODIFY_CURRENT_ENV
8503 CWB_POLICY_MODIFY_ENV_LIST
8504 CWB_SYSTEM_NOT_FOUND
8505 CWB_ENVIRONMENT_NOT_FOUND
8506 CWB_ENVIRONMENT_EXISTS
8507 CWB_SYSTEM_EXISTS
8508 CWB_NO_SYSTEMS_CONFIGURED
8580 CWB_CONFIGERR_RESERVED_START
8599 CWB_CONFIGERR_RESERVED_END

Automation Object return codes:

8600 CWB_INVALID_METHOD_PARM
8601 CWB_INVALID_PROPERTY_PARM
8602 CWB_INVALID_PROPERTY_VALUE
8603 CWB_OBJECT_NOT_INITIALIZED
8604 CWB_OBJECT_ALREADY_INITIALIZED
8605 CWB_INVALID_DQ_ORDER
8606 CWB_DATA_TRANSFER_REQUIRED
8607 CWB_UNSUPPORTED_XFER_REQUEST
8608 CWB_ASYNC_REQUEST_ACTIVE
8609 CWB_REQUEST_TIMED_OUT
8610 CWB_CANNOT_SET_PROP_NOW
8611 CWB_OBJ_STATE_NO_LONGER_VALID

WINSOCK return codes:

10024 CWB_TOO_MANY_OPEN_SOCKETS
10035 CWB_RESOURCE_TEMPORARILY_UNAVAILABLE
10038 CWB_SOCKET_OPERATION_ON_NON_SOCKET
10047 CWB_PROTOCOL_NOT_INSTALLED
10050 CWB_NETWORK_IS_DOWN
10051 CWB_NETWORK_IS_UNREACHABLE
10052 CWB_NETWORK_DROPPED_CONNECTION_ON_RESET
10053 CWB_SOFTWARE_CAUSED_CONNECTION_ABORT
10054 CWB_CONNECTION_RESET_BY_PEER
10055 CWB_NO_BUFFER_SPACE_AVAILABLE
10057 CWB_SOCKET_IS_NOT_CONNECTED
10058 CWB_CANNOT_SEND_AFTER_SOCKET_SHUTDOWN
10060 CWB_CONNECTION_TIMED_OUT
10061 CWB_CONNECTION_REFUSED
10064 CWB_HOST_IS_DOWN
10065 CWB_NO_ROUTE_TO_HOST
10091 CWB_NETWORK_SUBSYSTEM_IS_UNAVAILABLE
10092 CWB_WINSOCK_VERSION_NOT_SUPPORTED

11001 CWB_HOST_DEFINITELY_NOT_FOUND
The iSeries system name was not found during TCP/IP address lookup.

11002 CWB_HOST_NOT_FOUND_BUT_WE_ARE_NOT_SURE
The iSeries system name was not found during TCP/IP address lookup.

11004 CWB_VALID_NAME_BUT_NO_DATA_RECORD
The iSeries service name was not found in the local SERVICES file.

SSL return codes:

20001 CWB_SSL_ERROR_NO_CIPHERS
An I/O error occurred accessing the key database.

20002 CWB_SSL_ERROR_NO_CERTIFICATE
Open of the key database failed.

20004 CWB_SSL_ERROR_BAD_CERTIFICATE
Incorrect key database password.

20006 CWB_SSL_ERROR_UNSUPPORTED_CERTIFICATE_TYPE
Write to the key database failed.

20010 CWB_SSL_ERROR_IO
Certificate expired.

20011 CWB_SSL_ERROR_BAD_MESSAGE
Key already exists in the key database.

20012 CWB_SSL_ERROR_BAD_MAC
A bad message authentication code was received.

20013 CWB_SSL_ERROR_UNSUPPORTED

20014 CWB_SSL_ERROR_BAD_CERT_SIG
Duplicate key name in the key database.

20015 CWB_SSL_ERROR_BAD_CERT
Duplicate label in the key database.

20016 CWB_SSL_ERROR_BAD_PEER
Invalid password format for the key database.

20017 CWB_SSL_ERROR_PERMISSION_DENIED

20018 CWB_SSL_ERROR_SELF_SIGNED

20020 CWB_SSL_ERROR_BAD_MALLOC

20021 CWB_SSL_ERROR_BAD_STATE

20022 CWB_SSL_ERROR_SOCKET_CLOSED

20023 CWB_SSL_ERROR_INITIALIZATION_FAILED

20024 CWB_SSL_ERROR_HANDLE_CREATION_FAILED

20025 CWB_SSL_ERROR_BAD_DATE

20026 CWB_SSL_ERROR_BAD_KEY_LEN_FOR_EXPORT

20027 CWB_SSL_ERROR_NO_PRIVATE_KEY

20028 CWB_SSL_BAD_PARAMETER

20029 CWB_SSL_ERROR_INTERNAL

20030 CWB_SSL_ERROR_WOULD_BLOCK

20031 CWB_SSL_ERROR_LOAD_GSKLIB

20040 CWB_SSL_SOC_BAD_V2_CIPHER

20041 CWB_SSL_SOC_BAD_V3_CIPHER

20042 CWB_SSL_SOC_BAD_SEC_TYPE

20043 CWB_SSL_SOC_NO_READ_FUNCTION

20044 CWB_SSL_SOC_NO_WRITE_FUNCTION

20050 CWB_SSL_ERROR_NOT_SERVER

20051 CWB_SSL_ERROR_NOT_SSLV3

20052 CWB_SSL_ERROR_NOT_SSLV3_CLIENT

20099 CWB_SSL_ERROR_UNKNOWN_ERROR

20100 CWB_SSL_ERROR_BAD_BUFFER_SIZE

20101 CWB_SSL_ERROR_BAD_SSL_HANDLE

20102 CWB_SSL_ERROR_TIMEOUT

25001 CWB_SSL_KEYFILE_IO_ERROR

25002 CWB_SSL_KEYFILE_OPEN_FAILED

25003 CWB_SSL_KEYFILE_BAD_FORMAT

25004 CWB_SSL_KEYFILE_BAD_PASSWORD

25005 CWB_SSL_KEYFILE_BAD_MALLOC

25006 CWB_SSL_KEYFILE_NOTHING_TO_WRITE

25007 CWB_SSL_KEYFILE_WRITE_FAILED

25008 CWB_SSL_KEYFILE_NOT_FOUND

25009 CWB_SSL_KEYFILE_BAD_DNAME

25010 CWB_SSL_KEYFILE_BAD_KEY

25011 CWB_SSL_KEYFILE_KEY_EXISTS

25012 CWB_SSL_KEYFILE_BAD_LABEL
 25013 CWB_SSL_KEYFILE_DUPLICATE_NAME
 25014 CWB_SSL_KEYFILE_DUPLICATE_KEY
 25015 CWB_SSL_KEYFILE_DUPLICATE_LABEL
 25016 CWB_SSL_BAD_FORMAT_OR_INVALID_PW
 25098 CWB_SSL_WARNING_INVALID_SERVER_CERT
 25099 CWB_SSL_WARNING_INVALID_SERVER_PRIV_KEY
 25100 CWB_SSL_ERR_INIT_PARM_NOT_VALID
 25102 CWB_SSL_INIT_SEC_TYPE_NOT_VALID
 25103 CWB_SSL_INIT_V2_TIMEOUT_NOT_VALID
 25104 CWB_SSL_INIT_V3_TIMEOUT_NOT_VALID
 25105 CWB_SSL_KEYFILE_CERT_EXPIRED

iSeries Access for Windows component-specific return codes

- “Administration APIs return code”
- “Communications APIs return codes”
- “Database APIs return codes”
- “Data Queues APIs return codes” on page 25
- “Directory Update APIs return codes” on page 27
- “National language support APIs return codes” on page 27
- “iSeries Object APIs return codes” on page 28
- “Remote Command/Distributed Program Call APIs return codes” on page 28
- “Security APIs return codes” on page 29
- “Serviceability APIs return codes” on page 30
- “System Object Access APIs return codes” on page 30

Administration APIs return code:

6001 CWBAD_INVALID_COMPONENT_ID
 The component ID is invalid.

Communications APIs return codes:

6001 CWBCO_END_OF_LIST
 The end of system list has been reached. No system name was returned.

6002 CWBCO_DEFAULT_SYSTEM_NOT_DEFINED
 The setting for the default system has not been defined.

6003 CWBCO_DEFAULT_SYSTEM_NOT_CONFIGURED
 The default system is defined, but no connection to it is configured.

6004 CWBCO_SYSTEM_NOT_CONNECTED
 The specified system is not currently connected in the current process.

6005 CWBCO_SYSTEM_NOT_CONFIGURED
 The specified system is not currently configured.

6007 CWBCO_INTERNAL_ERROR
 Internal error.

6008 CWBCO_NO_SUCH_ENVIRONMENT
 The specified environment does not exist.

6009 CWB_TIMED_OUT
 The connect timeout value associated with the stem object expired before the connection attempt completed, so we stopped waiting.

Database APIs return codes:

6001 CWBDB_CANNOT_CONTACT_SERVER
 An error was encountered which prevented the Data Access server from being started.

6002 CWBDB_ATTRIBUTES_FAILURE
 An error was encountered during attempt to set the Data Access server attributes.

6003 CWBDB_SERVER_ALREADY_STARTED
 An attempt to start the Data Access server was made while a valid server was running. Stop the server before restarting it.

6004 CWBDB_INVALID_DRDA_PKG_SIZE
The valid submitted for the DRDA package size was invalid.

6005 CWBDB_REQUEST_MEMORY_ALLOCATION_FAILURE
A memory allocation attempt by a request handle failed.

6006 CWBDB_REQUEST_INVALID_CONVERSION
A Request handle failed in an attempt to convert data.

6007 CWBDB_SERVER_NOT_ACTIVE
The Data Access server is not started. It must be started before continuing.

6008 CWBDB_PARAMETER_ERROR
Attempt to set a parameter failed. Re-try. If error persists, there may be a lack of available memory.

6009 CWBDB_CLONE_CREATION_ERROR
Could not create a clone request.

6010 CWBDB_INVALID_DATA_FORMAT_FOR_CONNECTION
The data format object was not valid for this connection.

6011 CWBDB_DATA_FORMAT_IN_USE
The data format object is already being used by another request.

6012 CWBDB_INVALID_DATA_FORMAT_FOR_DATA
The data format object does not match the format of the data.

6013 CWBDB_STRING_ARG_TOO_LONG
The string provided was too long for the parameter.

6014 CWBDB_INVALID_INTERNAL_ARG
Invalid internally generated argument (not user supplied).

6015 CWBDB_INVALID_NUMERIC_ARG
Value of numeric argument is invalid.

6016 CWBDB_INVALID_ARG
Value of argument is invalid.

6017 CWBDB_STMT_NOT_SELECT
The statement provided was not a SELECT statement. This call requires a SELECT statement.

6018 CWBDB_STREAM_FETCH_NOT_COMPLETE
The connection is in stream fetch mode. Cannot perform desired operation until stream fetch has ended.

6019 CWBDB_STREAM_FETCH_NOT_ACTIVE
The connection is not in stream fetch mode and must be in order to perform the desired operation.

6020 CWBDB_MISSING_DATA_PROCESSOR
Pointer to data processor in request object is null.

6021 CWBDB_ILLEGAL_CLONE_REQUEST_TYPE
Cannot create a clone of an attributes request.

6022 CWBDB_UNSOLICITED_DATA
Data were received from the server, but none were requested.

6023 CWBDB_MISSING_DATA
Data were requested from the server, but not all were received.

6024 CWBDB_PARM_INVALID_BITSTREAM
Bitstream within a parameter is invalid.

6025 CWBDB_CONSISTENCY_TOKEN_ERROR
The data format used to interpret the data from the iSeries does not match the data returned.

6026 CWBDB_INVALID_FUNCTION
The function is invalid for this type of request.

6027 CWBDB_FORMAT_INVALID_ARG
A parameter value passed to the API was not valid.

6028 CWBDB_INVALID_COLUMN_POSITION
The column position passed to the API was not valid.

6029 CWBDB_INVALID_COLUMN_TYPE
The column type passed to the API was not valid.

6030 CWBDB_ROW_VECTOR_NOT_EMPTY
Invalid or corrupted format handle.

6031 CWBDB_ROW_VECTOR_EMPTY
Invalid or corrupted format handle.

6032 CWBDB_MEMORY_ALLOCATION_FAILURE
An error occurred while attempting to allocate memory.

6033 CWBDB_INVALID_CONVERSION
An invalid type conversion was attempted.

6034 CWBDB_DATASTREAM_TOO_SHORT

6035 CWBDB_SQL_WARNING The data stream received from the host was too short.
 The database server received a warning from an SQL operation.

6036 CWBDB_SQL_ERROR The database server received an error from an SQL operation.

6037 CWBDB_SQL_PARAMETER_WARNING The database server received a warning about a parameter used in an SQL operation.

6038 CWBDB_SQL_PARAMETER_ERROR The database server received an error about a parameter used in an SQL operation.

6039 CWBDB_LIST_SERVER_WARNING The database server returned a warning from a catalog operation.

6040 CWBDB_LIST_SERVER_ERROR The database server returned an error from a catalog operation.

6041 CWBDB_LIST_PARAMETER_WARNING The database server returned a warning about a parameter used in a catalog operation.

6042 CWBDB_LIST_PARAMETER_ERROR The database server returned an error about a parameter used in a catalog operation.

6043 CWBDB_NDB_FILE_SERVER_WARNING The database server returned a warning from a file processing operation.

6044 CWBDB_NDB_FILE_SERVER_ERROR The database server returned an error from a file processing operation.

6045 CWBDB_FILE_PARAMETER_WARNING The database server returned a warning about a parameter used in a file processing operation.

6046 CWBDB_FILE_PARAMETER_ERROR The database server returned an error about a parameter used in a file processing operation.

6047 CWBDB_GENERAL_SERVER_WARNING The database server returned a general warning.

6048 CWBDB_GENERAL_SERVER_ERROR The database server returned a general error.

6049 CWBDB_EXIT_PROGRAM_WARNING The database server returned a warning from an exit program.

6050 CWBDB_EXIT_PROGRAM_ERROR The database server returned an error from an exit program.

6051 CWBDB_DATA_BUFFER_TOO_SMALL Target data buffer is smaller than source buffer.

6052 CWBDB_NL_CONVERSION_ERROR Received error back from PiNConverter.

6053 CWBDB_COMMUNICATIONS_ERROR Received a communications error during processing.

6054 CWBDB_INVALID_ARG_API Value of argument is invalid - API level.

6055 CWBDB_MISSING_DATA_HANDLER Data handler not found in data handler list.

6056 CWBDB_REQUEST_DATASTREAM_NOT_VALID Invalid datastream in catalog request.

6057 CWBDB_SERVER_UNABLE Server incapable of performing desired function.

The following return codes are returned by the `cwbDB_StartServerDetailed` API:

6058 CWBDB_WORK_QUEUE_START_ERROR Unable to start server because of client work queue problem.

6059 CWBDB_WORK_QUEUE_CREATE_ERROR Unable to start server because of client work queue problem.

6060 CWBDB_INITIALIZATION_ERROR Unable to start server because of client initialization problem.

6061 CWBDB_SERVER_ATTRIBS_ERROR Unable to start server because of server attribute problem.

6062 CWBDB_CLIENT_LEVEL_ERROR

6063 CWBDB_CLIENT_LFC_ERROR Unable to start server because of set client level problem.
 Unable to start server because of set client language feature code problem.

6064 CWBDB_CLIENT_CCSID_ERROR Unable to start server because of set client CCSID problem.

6065 CWBDB_TRANSLATION_INDICATOR_ERROR Unable to start server because of set translation indicator error.

6066 CWBDB_RETURN_SERVER_ATTRIBS_ERROR Unable to start server because of return server attribute problem.

6067 CWBDB_SERVER_ATTRIBS_REQUEST Unable to start server because of missing server attributes request object.

6068 CWBDB_RETURN_ATTRIBS_ERROR Unable to start server because of return attribute problem.

6069 CWBDB_SERVER_ATTRIBS_MISSING Unable to start server because returned server attributes too short (missing data).

6070 CWBDB_SERVER_LFC_CONVERSION_ERROR Unable to start server because of data conversion error on server language feature code field of server attributes.

6071 CWBDB_SERVER_LEVEL_CONVERSION_ERROR Unable to start server because of data conversion error on server functional level field of server attributes.

6072 CWBDB_SERVER_LANGUAGE_TABLE_ERROR Unable to start server because of data conversion error on server language table ID field of server attributes.

6073 CWBDB_SERVER_LANGUAGE_LIBRARY_ERROR Unable to start server because of data conversion error on server language library ID field of server attributes.

6074 CWBDB_SERVER_LANGUAGE_ID_ERROR Unable to start server because of data conversion error on server language ID field of server attributes.

6075 CWBDB_COMM_DEQUEUE_ERROR Unable to start server because of communications error.

6076 CWBDB_COMM_ENQUEUE_ERROR Unable to start server because of communications error.

6077 CWBDB_UNSUPPORTED_COLUMN_TYPE An unsupported column type was found in the data.

6078 CWBDB_SERVER_IN_USE A connection to the database server for the given connection handle is already being used by another connection handle which was created with the same system object handle.

6099 CWBDB_LAST_STREAM_CHUNK Stream fetch complete.
 NOTE: Informational; not an error. There is no message or help text for this return code.

Data Queues APIs return codes:

6000 CWBDQ_INVALID_ATTRIBUTE_HANDLE Invalid attributes handle.

6001 CWBDQ_INVALID_DATA_HANDLE Invalid data handle.

6002 CWBDQ_INVALID_QUEUE_HANDLE Invalid queue handle.

6003 CWBDQ_INVALID_READ_HANDLE Invalid data queue read handle.

6004 CWBDQ_INVALID_QUEUE_LENGTH Invalid maximum record length for a data queue.

6005 CWBDQ_INVALID_KEY_LENGTH Invalid key length.

6006 CWBDQ_INVALID_ORDER Invalid queue order.

6007 CWBDQ_INVALID_AUTHORITY Invalid queue authority.

6008 CWBDQ_INVALID_QUEUE_TITLE
Queue title (description) is too long or cannot be converted.

6009 CWBDQ_BAD_QUEUE_NAME
Queue name is too long or cannot be converted.

6010 CWBDQ_BAD_LIBRARY_NAME
Library name is too long or cannot be converted.

6011 CWBDQ_BAD_SYSTEM_NAME
System name is too long or cannot be converted.

6012 CWBDQ_BAD_KEY_LENGTH
Length of key is not correct for this data queue or key length is greater than 0 for a LIFO or FIFO data queue.

6013 CWBDQ_BAD_DATA_LENGTH
Length of data is not correct for this data queue. Either the data length is zero or it is greater than the maximum allowed of 31744 bytes (64512 bytes for V4R5 and later versions of OS/400).
Note: The maximum allowed data length when connected to OS/400 V4R5M0 and later systems has been increased to 64512 bytes. When connected to earlier releases of OS/400, 64512 bytes of data may be written to a data queue, but the maximum length of data that may be read from a data queue is 31744 bytes.

6014 CWBDQ_INVALID_TIME
Wait time is not correct.

6015 CWBDQ_INVALID_SEARCH
Search order is not correct.

6016 CWBDQ_DATA_TRUNCATED
Returned data was truncated.

6017 CWBDQ_TIMED_OUT
Wait time has expired and no data has been returned.

6018 CWBDQ_REJECTED_USER_EXIT
Command rejected by user exit program.

6019 CWBDQ_USER_EXIT_ERROR
Error in user exit program or invalid number of exit programs.

6020 CWBDQ_LIBRARY_NOT_FOUND
Library not found on system.

6021 CWBDQ_QUEUE_NOT_FOUND
Queue not found on system.

6022 CWBDQ_NO_AUTHORITY
No authority to library or data queue.

6023 CWBDQ_DAMAGED_QUEUE
Data queue is in an unusable state.

6024 CWBDQ_QUEUE_EXISTS
Data queue already exists.

6025 CWBDQ_INVALID_MESSAGE_LENGTH
Invalid message length - exceeds queue maximum record length.

6026 CWBDQ_QUEUE_DESTROYED
Queue destroyed while waiting to read or peek a record.

6027 CWBDQ_NO_DATA
No data was received.

6028 CWBDQ_CANNOT_CONVERT
Data cannot be converted for this data queue. The data queue can be used but data cannot be converted between ASCII and EBCDIC. The convert flag on the data object will be ignored.

6029 CWBDQ_QUEUE_SYNTAX
Syntax of the data queue name is incorrect. Queue name must follow iSeries object syntax. First character must be alphabetic and all following characters alphanumeric.

6030 CWBDQ_LIBRARY_SYNTAX
Syntax of the library name is incorrect. Library name must follow iSeries object syntax. First character must be alphabetic and all

- 6031 CWBDQ_ADDRESS_NOT_SET
Address not set. The data object was not set with `cwbdQ_SetDataAddr()`, so the address cannot be retrieved. Use `cwbdQ_GetData()` instead of `cwbdQ_GetDataAddr()`.
- 6032 CWBDQ_HOST_ERROR
Host error occurred for which no return code is defined. See the error handle for the message text.
- 6033 CWBDQ_INVALID_SYSTEM_HANDLE
System handle is invalid.
- 6099 CWBDQ_UNEXPECTED_ERROR
Unexpected error.

Directory Update APIs return codes:

- 6000 CWBUP_ENTRY_NOT_FOUND
No update entry matched search value.
- 6001 CWBUP_SEARCH_POSITION_ERROR
Search starting position is not valid.
- 6002 CWBUP_PACKAGE_NOT_FOUND
The package file was not found.
- 6003 CWBUP_POSITION_INVALID
Position that is given is not in range.
- 6004 CWBUP_TOO_MANY_ENTRIES
The maximum number of update entries already exist. No more can be created.
- 6005 CWBUP_TOO_MANY_PACKAGES
Maximum number of package files already exists for this entry.
- 6006 CWBUP_STRING_TOO_LONG
The text string parameter passed in is longer than `CWBUP_MAX_LENGTH`.
- 6007 CWBUP_ENTRY_IS_LOCKED
Another application is currently changing the update entry list. No changes are allowed at this time.
- 6008 CWBUP_UNLOCK_WARNING
Application did not have the update entries locked.

National language support APIs return codes:

- 6101 CWBNL_ERR_CNV_UNSUPPORTED
An attempt was made to convert character data from a code page to another code page but this conversion is not supported.
- 6102 CWBNL_ERR_CNV_TBL_INVALID
A conversion table is in a format that is not recognized.
- 6103 CWBNL_ERR_CNV_TBL_MISSING
An attempt was made to use a conversion table, but the table was not found.
- 6104 CWBNL_ERR_CNV_ERR_GET
A code page conversion table was being retrieved from the server when an error occurred.
- 6105 CWBNL_ERR_CNV_ERR_COMM
A code page conversion table was being retrieved from the server when a communications error occurred.
- 6106 CWBNL_ERR_CNV_ERR_SERVER
A code page conversion table was being retrieved from the server when a server error occurred.
- 6107 CWBNL_ERR_CNV_ERR_STATUS
While converting character data from one code page to another, some untranslatable characters were encountered.
- 6108 CWBNL_ERROR_CONVERSION_INCOMPLETE_MULTIBYTE_INPUT_CHARACTER
While converting character data an incomplete multibyte character was found.
- 6109 CWBNL_ERR_CNV_INVALID_SISO_STATUS
The SISO parameter is incorrect.
- 6110 CWBNL_ERR_CNV_INVALID_PAD_LENGTH
The pad length parameter is incorrect.

The following return codes are for language APIs:

6201	CWBNL_ERR_STR_TBL_INVALID	Message file not in a recognized format. It has been corrupted.
6202	CWBNL_ERR_STR_TBL_MISSING	Message file could not be found.
6203	CWBNL_ERR_STR_NOT_FOUND	The message file is missing a message.
6204	CWBNL_ERR_NLV_NO_CONFIG	The language configuration is missing.
6205	CWBNL_ERR_NLV_NO_SUBDIR	The language subdirectory is missing.
6206	CWBNL_DEFAULT_HOST_CCSID_USED	A default server CCSID (500) is used.

The following return codes are for locale APIs:

6301	CWBNL_ERR_LOC_TBL_INVALID
6302	CWBNL_ERR_LOC_TBL_MISSING
6303	CWBNL_ERR_LOC_NO_CONFIG
6304	CWBNL_ERR_LOC_NO_LOCPATH

iSeries Object APIs return codes:

6000	CWBOBJ_RC_HOST_ERROR	Host error occurred. Text may be in errorHandle.
6001	CWBOBJ_RC_INVALID_TYPE	Incorrect object type.
6002	CWBOBJ_RC_INVALID_KEY	Incorrect key.
6003	CWBOBJ_RC_INVALID_INDEX	Bad index to list.
6004	CWBOBJ_RC_LIST_OPEN	The list is already opened.
6005	CWBOBJ_RC_LIST_NOT_OPEN	The list has not been opened.
6006	CWBOBJ_RC_SEEKOUTOFRANGE	Seek offset is out of range.
6007	CWBOBJ_RC_SPLFNOTOPEN	Spooled file has not been opened.
6007	CWBOBJ_RC_RSCNOTOPEN	Resource has not been opened.
6008	CWBOBJ_RC_SPLFENDOFFILE	End of file was reached.
6008	CWBOBJ_RC_ENDOFFILE	End of file was reached.
6009	CWBOBJ_RC_SPLFNOMESSAGE	The spooled file is not waiting on a message.
6010	CWBOBJ_RC_KEY_NOT_FOUND	The parameter list does not contain the specified key.
6011	CWBOBJ_RC_NO_EXIT_PGM	No exit program registered.
6012	CWBOBJ_RC_NOHOSTSUPPORT	Host does not support function.

Remote Command/Distributed Program Call APIs return codes:

6000	CWBRC_INVALID_SYSTEM_HANDLE	Invalid system handle.
6001	CWBRC_INVALID_PROGRAM	Invalid program handle.
6002	CWBRC_SYSTEM_NAME	System name is too long or cannot be converted.
6003	CWBRC_COMMAND_STRING	Command string is too long or cannot be converted.
6004	CWBRC_PROGRAM_NAME	Program name is too long or cannot be converted.
6005	CWBRC_LIBRARY_NAME	

6006 CWBRC_INVALID_TYPE
 Library name is too long or cannot be converted.
 Invalid parameter type specified.
 6007 CWBRC_INVALID_PARM_LENGTH
 Invalid parameter length.
 6008 CWBRC_INVALID_PARM
 Invalid parameter specified.
 6009 CWBRC_TOO_MANY_PARMS
 Attempt to add more than 25 parameters to a program.
 6010 CWBRC_INDEX_RANGE_ERROR
 Index is out of range for this program.
 6011 CWBRC_REJECTED_USER_EXIT
 Command rejected by user exit program.
 6012 CWBRC_USER_EXIT_ERROR
 Error in user exit program.
 6013 CWBRC_COMMAND_FAILED
 Command failed.
 6014 CWBRC_PROGRAM_NOT_FOUND
 Program not found or could not be accessed.
 6015 CWBRC_PROGRAM_ERROR
 Error occurred when calling the program.
 6016 CWBRC_COMMAND_TOO_LONG
 Command string is longer than the maximum of 6000 characters.
 6099 CWBRC_UNEXPECTED_ERROR
 Unexpected error.

Security APIs return codes:

6000 CWBSY_UNKNOWN_USERID
 User ID does not exist.
 6002 CWBSY_WRONG_PASSWORD
 Password is not correct for specified user ID.
 6003 CWBSY_PASSWORD_EXPIRED
 Password has expired.
 6004 CWBSY_INVALID_PASSWORD
 One or more characters in the password are not valid or the password is too long.
 6007 CWBSY_GENERAL_SECURITY_ERROR
 A general security error occurred. The user profile does not have a password or the password validation program found an error in the password.
 6009 CWBSY_INVALID_PROFILE
 The AS/400 user profile is not valid.
 6011 CWBSY_USER_PROFILE_DISABLED
 The iSeries user profile (user ID) has been set to disabled.
 6013 CWBSY_USER_CANCELLED
 The user cancelled from the user ID/password prompt.
 6015 CWBSY_INVALID_USERID
 One or more characters in the user ID is not valid or the user ID is too long.
 6016 CWBSY_UNKNOWN_SYSTEM
 The system specified is unknown.
 6019 CWBSY_TP_NOT_VALID
 The PC could not validate the iSeries security server. This could indicate tampering with the IBM supplied security server program on the iSeries.
 6022 CWBSY_NOT_LOGGED_ON
 There is no user currently logged on for the specified system.
 6025 CWBSY_SYSTEM_NOT_CONFIGURED
 The system specified in the security object has not been configured.
 6026 CWBSY_NOT_VERIFIED
 The user ID and password defined in the object has not yet been verified. You must verify using cwbsy_VerifyUserIDPw API.
 6255 CWBSY_INTERNAL_ERROR
 Internal error. Contact IBM Service.

The following return codes are for change password APIs:

- 6257 CWBSY_PWD_TOO_LONG
The new password contains too many characters. The maximum number of characters allowed is defined by the iSeries system value, QPWDMAXLEN.
- 6258 CWBSY_PWD_TOO_SHORT
The new password does not contain enough characters. The minimum number of characters allowed is defined by the iSeries system value, QPWDMINLEN.
- 6259 CWBSY_PWD_REPEAT_CHARACTER
The new password contains a character used more than once. The iSeries configuration (system value QPWDLMTREP) does not allow passwords to contain a repeat character.
- 6260 CWBSY_PWD_ADJACENT_DIGITS
The new password contains two numbers next to each other. The iSeries configuration (system value QPWDLMTAJC) does not allow passwords to contain consecutive digits.
- 6261 CWBSY_PWD_CONSECUTIVE_CHARS
The new password contains a character repeated consecutively. The iSeries configuration (system value QPWDLMTREP) does not allow a password to contain a character repeated consecutively.
- 6262 CWBSY_PWD_PREVIOUSLY_USED
The new password matches a previously used password. The iSeries configuration (system value QPWDRQDDIF) requires new passwords to be different than any previous password.
- 6263 CWBSY_PWD_DISALLOWED_CHAR
The new password uses an installation disallowed character. iSeries configuration (system value QPWDLMTCHR) restricts certain characters from being used in new passwords.
- 6264 CWBSY_PWD_NEED_NUMERIC
The new password must contain a number. The iSeries configuration (system value QPWDRQDDGT) requires new passwords contain one or more numeric digits.
- 6266 CWBSY_PWD_MATCHES_OLD
The new password matches an old password in one or more character positions. The AS/400 configuration (system value QPWDPOSDIF) does not allow the same character to be in the same position as a previous password.
- 6267 CWBSY_PWD_NOT_ALLOWED
The password was rejected.
- 6268 CWBSY_PWD_MATCHES_USERID
The password matches the user ID.
- 6269 CWBSY_PWD_PRE_V3
The old password was created on a pre-V3 system which used a different encryption technique. Password must be changed manually on the AS/400.
- 6270 CWBSY_LAST_INVALID_PASSWORD
The next invalid will disable the user profile.

Serviceability APIs return codes:

- 6000 CWBSV_INVALID_FILE_TYPE
Unusable file type passed-in.
- 6001 CWBSV_INVALID_RECORD_TYPE
Unusable record type passed-in.
- 6002 CWBSV_INVALID_EVENT_TYPE
Unusable event type detected.
- 6003 CWBSV_NO_ERROR_MESSAGES
No error messages associated with error handle.
- 6004 CWBSV_ATTRIBUTE_NOT_SET
Attribute not set in current message.
- 6005 CWBSV_INVALID_MSG_CLASS
Unusable message class passed-in.

System Object Access APIs return codes:

- 0 CWBSO_NO_ERROR
No error occurred.

```

1 CWBSO_ERROR_OCCURRED
    An error occurred. Use error handle for more information.
2 CWBSO_LOW_MEMORY
    Not enough memory is available for the request.
3 CWBSO_BAD_LISTTYPE
    The value specified for type of list is not valid.
4 CWBSO_BAD_HANDLE
    The handle specified is not valid.
5 CWBSO_BAD_LIST_HANDLE
    The list handle specified is not valid.
6 CWBSO_BAD_OBJ_HANDLE
    The object handle specified is not valid.
7 CWBSO_BAD_PARMOBJ_HANDLE
    The parameter object handle specified is not valid.
8 CWBSO_BAD_ERR_HANDLE
    The error handle specified is not valid.
9 CWBSO_BAD_LIST_POSITION
    The position in list specified does not exist.
10 CWBSO_BAD_ACTION_ID
    An action ID specified is not valid for the type of list.
11 CWBSO_NOT_ALLOWED_NOW
    The action requested is not allowed at this time.
12 CWBSO_BAD_INCLUDE_ID
    The filter ID specified is not valid for this list.
13 CWBSO_DISP_MSG_FAILED
    The request to display the message failed.
14 CWBSO_GET_MSG_FAILED
    The error message text could not be retrieved.
15 CWBSO_BAD_SORT_ID
    A sort ID specified is not valid for the type of list.
16 CWBSO_INTERNAL_ERROR
    An internal processing error occurred.
17 CWBSO_NO_ERROR_MESSAGE
    The error handle specified contains no error message.
18 CWBSO_BAD_ATTRIBUTE_ID
    The attribute key is not valid for this object.
19 CWBSO_BAD_TITLE
    The title specified is not valid.
20 CWBSO_BAD_FILTER_VALUE
    The filter value specified is not valid.
21 CWBSO_BAD_PROFILE_NAME
    The profile name specified is not valid.
22 CWBSO_DISPLAY_FAILED
    The window could not be created.
23 CWBSO_SORT_NOT_ALLOWED
    Sorting is not allowed for this type of list.
24 CWBSO_CANNOT_CHANGE_ATTR
    Attribute is not changeable at this time.
25 CWBSO_CANNOT_READ_PROFILE
    Cannot read from the specified profile file.
26 CWBSO_CANNOT_WRITE_PROFILE
    Cannot write to the specified profile file.
27 CWBSO_BAD_SYSTEM_NAME
    The system name specified is not a valid iSeries system name.
28 CWBSO_SYSTEM_NAME_DEFAULTED
    No system name was specified on the "CWBSO_CreateListHandle" call
    for the list.
29 CWBSO_BAD_FILTER_ID
    The filter ID specified is not valid for the type of list.

```

iSeries Access for Windows Administration APIs

iSeries Access for Windows Administration APIs provide functions that access information about the iSeries Access for Windows code that is installed on the PC. Administration APIs allow you to determine:

- The version and service level of iSeries Access for Windows
- The install status of individual components

- The install status of iSeries Navigator plug-ins

iSeries Access for Windows Administration APIs required files:

Header file	Import library	Dynamic Link Library
cwbad.h	cwbapi.lib	cwbad.dll

Programmer’s Toolkit:

The Programmer’s Toolkit provides Administration APIs documentation, access to the cwbad.h header file, and links to sample programs. To access this information, open the Programmer’s Toolkit and select **Client Information** —> **C/C++ APIs**.

iSeries Access for Windows Administration APIs topics:

- **iSeries Access for Windows Administration APIs listing**
- “Example: Administration APIs” on page 40
- “Administration APIs return code” on page 22

Related topics:

- “iSeries system name formats for ODBC Connection APIs” on page 12
- “OEM, ANSI, and Unicode considerations” on page 12

Administration APIs listing

- cwAD_GetClientVersion
- cwAD_GetProductFixLevel
- cwAD_IsComponentInstalled
- cwAD_IsOpNavPluginInstalled

cwbAD_GetClientVersion

Purpose: Get the version of the iSeries Access for Windows product that currently is installed on a PC.

Syntax:

```
unsigned int CWB_ENTRY cwbAD_GetClientVersion(  
                                     unsigned long *version  
                                     unsigned long *release  
                                     unsigned long *modificationLevel);
```

Parameters:

unsigned long *version - output

Pointer to a buffer where the version level of the iSeries Access for Windows product is returned.

unsigned long *release - output

Pointer to a buffer where the release level of the iSeries Access for Windows product is returned.

unsigned long *modificationLevel - output

Pointer to a buffer where the modification level of the iSeries Access for Windows product is returned.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

One or more pointer parameters are null.

Usage: If the return code is not CWB_OK, the values in version, release, and modificationLevel are meaningless.

cwbAD_GetProductFixLevel

Purpose: Returns the current fix level of iSeries Access for Windows.

Syntax:

```
unsigned int CWB_ENTRY cwbAD_GetProductFixLevel(  
    char *szBuffer  
    unsigned long *ulBufLen);
```

Parameters:

char *szBuffer - output

Buffer into which the product fix level string will be written.

unsigned long * ulBufLen - input/output

Size of szBuffer, including space for the NULL terminator. On output, will contain the length of the fix level string, including the terminating NULL.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Buffer overflow. The required length is returned in ulBufLen.

CWB_INVALID_POINTER

Invalid pointer.

Usage: Returns the fix level of the iSeries Access for Windows product. Returns an empty string if fixes have not been applied.

cwbAD_IsComponentInstalled

Purpose: Indicates whether a specific iSeries Access for Windows component is installed.

Syntax:

```
unsigned long CWB_ENTRY cwbAD_IsComponentInstalled(  
                unsigned long    ulComponentID,  
                cwb_Boolean      *bIndicator);
```

Parameters:

unsigned long ulComponentID - input

Must be set to one of the following component IDs:

CWBAD_COMP_SSL

Secure Sockets Layer

CWBAD_COMP_SSL_128_BIT

Secure Sockets Layer 128 bit

CWBAD_COMP_SSL_56_BIT

Secure Sockets Layer 56 bit

CWBAD_COMP_SSL_40_BIT

Secure Sockets Layer 40 bit

CWB_COMP_BASESUPPORT

iSeries Access for Windows required programs

CWBAD_COMP_OPTIONAL_COMPS

iSeries Access for Windows Optional Components

CWBAD_COMP_DIRECTORYUPDATE

Directory Update

CWBAD_COMP_IRC

Incoming Remote Command

CWBAD_COMP_MAPI

MAPI

CWBAD_COMP_OUG

User's Guide

CWBAD_COMP_OPNAV

iSeries Navigator

CWBAD_COMP_DATA_ACCESS

Data Access

CWBAD_COMP_DATA_TRANSFER

Data Transfer

CWBAD_COMP_DT_BASESUPPORT

Data Transfer Base Support

CWBAD_COMP_DT_EXCEL_ADDIN

Data Transfer Excel Add-in

CWBAD_COMP_DT_WK4SUPPORT

Data Transfer WK4 file support

CWBAD_COMP_ODBC

ODBC

CWBAD_COMP_OLEDB

OLE DB Provider

CWBAD_COMP_AFP_VIEWER

AFP™ Workbench Viewer

CWBAD_COMP_JAVA_TOOLBOX

Java Toolbox

CWBAD_COMP_PC5250

PC5250 Display and Printer Emulator

PC5250 Display and Printer Emulator subcomponents:

CWBAD_COMP_PC5250_BASE_KOREAN
CWBAD_COMP_PC5250_PDFPDT_KOREAN
CWBAD_COMP_PC5250_BASE_SIMPCHIN
CWBAD_COMP_PC5250_PDFPDT_SIMPCHIN
CWBAD_COMP_PC5250_BASE_TRADCHIN
CWBAD_COMP_PC5250_PDFPDT_TRADCHIN
CWBAD_COMP_PC5250_BASE_STANDARD
CWBAD_COMP_PC5250_PDFPDT_STANDAR
CWBAD_COMP_PC5250_FONT_ARABIC
CWBAD_COMP_PC5250_FONT_BALTIC
CWBAD_COMP_PC5250_FONT_LATIN2
CWBAD_COMP_PC5250_FONT_CYRILLIC
CWBAD_COMP_PC5250_FONT_GREEK
CWBAD_COMP_PC5250_FONT_HEBREW
CWBAD_COMP_PC5250_FONT_LAO
CWBAD_COMP_PC5250_FONT_THAI
CWBAD_COMP_PC5250_FONT_TURKISH
CWBAD_COMP_PC5250_FONT_VIET

CWBAD_COMP_PRINTERDRIVERS

Printer Drivers

CWBAD_COMP_AFP_DRIVER

AFP printer driver

CWBAD_COMP_SCS_DRIVER

SCS printer driver

CWBAD_COMP_OP_CONSOLE

Operations Console

CWBAD_COMP_TOOLKIT

Programmer's Toolkit

CWBAD_COMP_TOOLKIT_BASE

Headers, Libraries, and Documentation

CWBAD_COMP_TOOLKIT_VBW

Visual Basic Wizard

CWBAD_COMP_EZSETUP

EZ Setup

CWBAD_COMP_TOOLKIT_JAVA_TOOLS

Programmer's Toolkit Tools for Java

CWBAD_COMP_SCREEN_CUSTOMIZER_ENABLER

Screen Customizer Enabler

CWBAD_COMP_OPNAV_BASESUPPORT

iSeries Navigator Base Support

CWBAD_COMP_OPNAV_BASE_OPS

iSeries Navigator Basic Operations

CWBAD_COMP_OPNAV_JOB_MGMT

iSeries Navigator Job Management

CWBAD_COMP_OPNAV_SYS_CFG

iSeries Navigator System Configuration

CWBAD_COMP_OPNAV_NETWORK

iSeries Navigator Networks

CWBAD_COMP_OPNAV_SECURITY

iSeries Navigator Security

CWBAD_COMP_OPNAV_USERS_GROUPS

iSeries Navigator Users and Groups

CWBAD_COMP_OPNAV_DATABASE

iSeries Navigator Database

CWBAD_COMP_OPNAV_MULTIMEDIA

iSeries Navigator Multimedia

CWBAD_COMP_OPNAV_BACKUP

iSeries Navigator Backup

CWBAD_COMP_OPNAV_APP_DEV

iSeries Navigator Application Development

CWBAD_COMP_OPNAV_APP_ADMIN

iSeries Navigator Application Administration

CWBAD_COMP_OPNAV_FILE_SYSTEMS

iSeries Navigator File Systems

CWBAD_COMP_OPNAV_MGMT_CENTRAL

iSeries Navigator Management Central

CWBAD_COMP_OPNAV_MGMT_COMMANDS

iSeries Navigator Management Central - Commands

CWBAD_COMP_OPNAV_MGMT_PACK_PROD

iSeries Navigator Management Central - Packages and Products

CWBAD_COMP_OPNAV_MGMT_MONITORS

iSeries Navigator Management Central - Monitors

CWBAD_COMP_OPNAV_LOGICAL_SYS

iSeries Navigator Logical Systems

CWBAD_COMP_OPNAV_ADV_FUNC PRES

iSeries Navigator Advanced Function Presentation

cwb_Boolean *bIndicator - output

Will contain CWB_TRUE if the component is installed. Will return CWB_FALSE if the component is not installed. Will not be set if an error occurs.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Invalid pointer.

CWB_INVALID_COMPONENT_ID

The component ID is invalid for this release.

cwbAD_IsOpNavPluginInstalled

Purpose: Indicates whether a specific iSeries Navigator plug-in is installed.

Syntax:

```
unsigned long CWB_ENTRY cwbAD_IsOpNavPluginInstalled(  
    const char    *szPluginName,  
    cwb_Boolean   *bIndicator);
```

Parameters:

const char* szPluginName - input

Pointer to a null-terminated string that contains the name of the plug-in.

cwb_Boolean *bIndicator - output

Will contain CWB_TRUE if the plug-in is installed. Will return CWB_FALSE if the component is not installed. Will not be set if an error occurs.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

One of the pointer parameters is NULL.

Usage: If the return value is not CWB_OK, the value in bIndicator is meaningless.

Example: Administration APIs

This example demonstrates how an application might use iSeries Access for Windows Administration APIs. In this example, the APIs are used to get and display:

- The current iSeries Access for Windows Version/Release/Modification level
- The current service pack (fix) level
- The components that currently are installed on the PC

The user then is allowed to enter iSeries Navigator plug-in names, and is informed whether the plug-in is installed.

Usage notes:

Include cwbad.h *

Link with cwbapi.lib

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
#include "cwbad.h"
```

```
/*
```

```
* This is the highest numbered component ID we know about (it is  
* the ID of the last component defined in cwbad.h).
```

```
*/
```

```
#define LAST_COMPID_WE_KNOW_ABOUT      (CWBAD_COMP_SSL_40_BIT)
```

```
/*
```

```
* Array of component names, taken from comments for component IDs  
* in cwbad.h, so we can display human-readable component descriptions.  
* In the compDescr array, the component ID for a component must match  
* the index in the array of that component's description.
```

```
*
```

```
* For a blank or unknown component name, we provide a string to display  
* an indication that the component ID is unknown, and what that ID is.
```

```
*/
```

```
static char* compDescr[ LAST_COMPID_WE_KNOW_ABOUT + 1 ] = {  
    "", // #0 is not used  
    "Required programs",  
    "Optional Components",  
    "Directory Update",  
    "Incoming Remote Command",  
    "", // not used,  
    "Online User's Guide",  
    "iSeries Navigator",  
    "Data Access",  
    "Data Transfer",  
    "Data Transfer Base Support",  
    "Data Transfer Excel Add-in",  
    "Data Transfer WK4 file support",  
    "ODBC",  
    "OLE DB Provider",  
    "AFP Workbench Viewer",  
    "iSeries Java Toolbox",  
    "5250 Display and Printer Emulator",  
    "Printer Drivers",  
    "AFP printer driver",  
    "SCS printer driver",  
    "iSeries Operations Console",  
    "iSeries Access Programmer's Toolkit",  
    "Headers, Libraries, and Documentation",  
};
```

```

"Visual Basic Wizards",
"EZ Setup",
"Java Toolkit"
"Screen customizer" // #27
" ", " ", " ", " ", " ", " ", //----- #28-29
" ", " ", " ", " ", " ", // #30-34
" ", " ", " ", " ", " ", // #35-39
" ", " ", " ", " ", " ", // #40-44
" ", " ", " ", " ", " ", // #45-49
" ", " ", " ", " ", " ", // not #50-54
" ", " ", " ", " ", " ", // #55-59
" ", " ", " ", " ", " ", // #60-64
" ", " ", " ", " ", " ", // #65-69
" ", " ", " ", " ", " ", // used #70-74
" ", " ", " ", " ", " ", // #75-79
" ", " ", " ", " ", " ", // #80-84
" ", " ", " ", " ", " ", // #85-89
" ", " ", " ", " ", " ", // #90-94
" ", " ", " ", " ", " ", //----- #95-99
"iSeries Navigator Base Support", // #100
"iSeries Navigator Basic Operations",
"iSeries Navigator Job Management",
"iSeries Navigator System Configuration",
"iSeries Navigator Networks",
"iSeries Navigator Security",
"iSeries Navigator Users and Groups",
"iSeries Navigator Database",
"iSeries Navigator Multimedia",
"iSeries Navigator Backup",
"iSeries Navigator Application Development",
"iSeries Navigator Application Administrat",
"iSeries Navigator File Systems",
"iSeries Navigator Management Central",
"iSeries Navigator Management Central - Commands",
"iSeries Navigator Management Central - Packages and Products",
"iSeries Navigator Logical Systems",
"iSeries Navigator Advanced Function Presentation",
" ", //----- #119
" ", " ", " ", " ", " ", // not #120-124
" ", " ", " ", " ", " ", // #125-129
" ", " ", " ", " ", " ", // #130-134
" ", " ", " ", " ", " ", // used #135-139
" ", " ", " ", " ", " ", // #140-144
" ", " ", " ", " ", " ", //----- #145-149
"PC5250: BASE_KOREAN", // #150
"PC5250: PDFPDT_KOREAN",
"PC5250: BASE_SIMPCHIN",
"PC5250: PDFPDT_SIMPCHIN",
"PC5250: BASE_TRADCHIN",
"PC5250: PDFPDT_TRADCHIN",
"PC5250: BASE_STANDARD",
"PC5250: PDFPDT_STANDARD",
"PC5250: FONT_ARABIC",
"PC5250: FONT_BALTIC",
"PC5250: FONT_LATIN2",
"PC5250: FONT_CYRILLIC",
"PC5250: FONT_GREEK",
"PC5250: FONT_HEBREW",
"PC5250: FONT_LAO",
"PC5250: FONT_THAI",
"PC5250: FONT_TURKISH",
"PC5250: FONT_VIET",
" ", " ", //----- #168-169

```

```

        "", "", "", "", "", // #170-174
        "", "", "", "", "", // not #175-179
        "", "", "", "", "", // #180-184
        "", "", "", "", "", // used #185-189
        "", "", "", "", "", // #190-194
        "", "", "", "", "", //----- #195-199
        "Secure Sockets Layer (SSL)",
        "SSL 128-bit subcomponent",
        "SSL 56-bit subcomponent",
        "SSL 40-bit subcomponent" } ; // last one defined
static char unknownComp[] = "unknown, ID= ";
static char* pInsertID = &("unknownComp[12] ); // insert ID here!

```

```

/*****
 * Show the iSeries Access for Windows Version/Release/Modification level
 *****/

```

```

void showCA_VRM()
{
    ULONG caVer, caRel, caMod;
    UINT rc;
    char fixlevelBuf[ MAX_PATH ];
    ULONG fixlevelBufLen = sizeof( fixlevelBuf );

    printf( "iSeries Access level installed:\n\n" );

    rc = cwBAD_GetClientVersion( &caVer, &caRel, &caMod );
    if ( rc != CWB_OK )
    {
        printf( " Error %u occurred when calling cwBAD_GetClientVersion()\n\n",
            rc );
    }
    else
    {
        printf( " Version %lu, Release %lu, Modification %lu\n\n",
            caVer, caRel, caMod );

        printf( "iSeries Access service pack level installed:\n\n" );
        rc = cwBAD_GetProductFixLevel( fixlevelBuf, &fixlevelBufLen );
        if ( rc != CWB_OK )
        {
            printf( " Error %u occurred when calling "
                "cwBAD_GetProduceFixLevel()\n\n", rc );
        }
        else if ( fixlevelBuf[0] == '\0' ) // empty, no service packs applied
        {
            printf( " None\n\n" );
        }
        else
        {
            printf( " %s\n\n", fixlevelBuf );
        }
    }
}

```

```

/*****
 * Call iSeries Access for Windows API to determine if the component is installed,
 * and pass back:
 * NULL if the component is not installed or an error occurs,

```



```

*           OR
*   A string indicating the component name is unknown if the
*   component ID is higher than we know about OR the component
*   description is blank,
*           OR
*   The human-readable component description if we know it.
*****/
char* isCompInstalled( ULONG compID )
{
    cwb_Boolean bIsInstalled;
    char*       pCompName;

    UINT rc = cwbAD_IsComponentInstalled( compID, &bIsInstalled; );

    /*
     * Case 1: Error OR component not installed, return NULL to
     * indicate not installed.
     */
    if ( ( rc != CWB_OK ) || ( bIsInstalled == CWB_FALSE ) )
    {
        pCompName = NULL;
    }

    /*
     * Case 2: Component IS installed, but we do not know its name,
     * return component name unknown string.
     */
    else if ( ( compID > LAST_COMPID_WE_KNOW_ABOUT ) ||
              ( compDescr[ compID ][ 0 ] == '\0' ) )
    {
        pCompName = unknownComp;
        sprintf( pInsertID, "%lu", compID );
    }

    /*
     * Case 3: Component IS installed, we have a name, return it
     */
    else
    {
        pCompName = compDescr[ compID ];
    }

    return pCompName;
}

/*****
 * List the iSeries Access for Windows components that currently are installed.
 *****/
void showCA_CompInstalled()
{
    ULONG compID;
    char* compName;

    printf( "iSeries Access components installed:\n\n" );

    /*
     * Try all components we know about, plus a bunch more in case some
     * have been added (via service pack).
     */
    for ( compID = 0;
          compID &lt; (LAST_COMPID_WE_KNOW_ABOUT + 50);

```

```

        compID++ )
    {
        compName = isCompInstalled( compID );
        if ( compName != NULL )
        {
            printf( "   %s\n", compName );
        }
    }

    printf( "\n" );
}

```

```

/*****
 * MAIN PROGRAM BODY
 *****/
void main(void)
{
    UINT          rc;
    char          pluginName[ MAX_PATH ];
    cwb_Boolean   bPluginInstalled;

    printf( "=====\n" );
    printf( "iSeries Access What's Installed Reporter\n" );
    printf( "=====\n\n" );

    showCA_VRM();
    showCA_CompInstalled();

    /*
     * Allow user to ask by name what plug-ins are installed.
     */
    while ( TRUE ) /* REMINDER: requires a break to exit the loop! */
    {
        printf( "Enter plug-in to check for, or DONE to quit:\n" );
        gets( pluginName );
        if ( strcmp( pluginName, "DONE" ) == 0 )
        {
            break; /* exit from the while loop, we are DONE at user's request */
        }

        rc = cwbAD_IsOpNavPluginInstalled( pluginName, &bPluginInstalled );
        if ( rc == CWB_OK )
        {
            if ( bPluginInstalled == CWB_TRUE )
            {
                printf( "The plug-in '%s' is installed.\n\n", pluginName );
            }
            else
            {
                printf( "The plug-in '%s' is NOT installed.\n\n", pluginName );
            }
        }
        else
        {
            printf(
                "Error %u occurred when calling cwbAD_IsOpNavPluginInstalled.\n\n",
                rc );
        }
    }
}

```

```

} // end while (TRUE)

printf( "\nEnd of program.\n\n" );
}

```

iSeries Access for Windows Communications and Security APIs

The iSeries Access for Windows Communications and Security topic shows you how to use iSeries Access for Windows application programming interfaces (APIs) to:

- Get, use, and delete an iSeries **system object**. Various iSeries Access for Windows APIs require a system object. It holds information about connecting to, and validating security (user ID, password, and signon date and time) on, an iSeries system. For more information, see “System object attributes” and “System object attributes listing” on page 46.
- Obtain information about environments and connections that are configured in the **system list** when you use iSeries Access for Windows. The system list is a list of all currently configured environments, and of systems within those environments. The system list is stored and managed “per user,” and is not available to other users.

Note: It is not necessary for you to explicitly configure new systems to add them to the system list. They are added automatically when you connect to a new system.

iSeries Access for Windows Communications and Security APIs required files:

Header file		Import library	Dynamic Link Library
System object APIs	System list APIs	cwbapi.lib	cwbcod.dll
cwbcosys.h	cwbco.h		

Programmer’s Toolkit:

The Programmer’s Toolkit provides Communications and Security documentation, access to the cwbcod.h and cwbcosys.h header files, and links to sample programs. To access this information, open the Programmer’s Toolkit and select **Communications and Security** → **C/C++ APIs**.

iSeries Access for Windows Communications and Security topics:

- **iSeries Access for Windows Communications and Security system object APIs listing**
- **iSeries Access for Windows Communications system list APIs listing**
- “Example: Using iSeries Access for Windows communications APIs” on page 132
- “Communications APIs return codes” on page 22
- “Security APIs return codes” on page 29
- “Global iSeries Access return codes” on page 18

Related topics:

- “iSeries system name formats for ODBC Connection APIs” on page 12
- “OEM, ANSI, and Unicode considerations” on page 12

System object attributes

System object attributes affect the behavior of signing on and communicating with the iSeries system that the system object represents.

Most attributes can be changed until a successful signon has occurred (either as the result of a successful call to “cwbCO_Signon” on page 102 or to “cwbCO_Connect” on page 59). After the signon has taken place successfully, calling the API that tries to change the value of such an attribute will fail with return code CWB_INV_AFTER_SIGNON. The only two attributes that can be changed after a successful signon are the Window Handle and Connect Timeout.

Some values and the ability to change them may be controlled via **policies**. Policies are controls that a systems administrator can set up to mandate default attribute values, and to prohibit changes to attributes. The default values that are specified in the **System object attributes listing** topic (link below) are used under the following conditions:

- If policies do not specify or suggest different values
- If a value for such an attribute has not been configured explicitly for the iSeries system in the system list

If an attribute's default value may be set by policy, this also is noted. If changing an attribute's value can be prohibited by policy, then:

- An API is provided to check for the attribute's modifiability.
- A specific return code is provided by the attribute's set method if the set fails because of such a policy.

To view a listing of system object attributes:

See "System object attributes listing"

System object attributes listing

Following is a list of system object attributes. It includes descriptions, requirements, and considerations. Also listed with each attribute are:

- The APIs that you can use to get and to set it
- What its default value is when the system object is created

Note: The attributes' settings apply **ONLY** to the system object for which they are set, **NOT** to any other system objects, even if other system objects have the same iSeries system name.

iSeries system name:

The iSeries system with which to communicate and use by way of this instance of a system object. This can be set only at the time `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike` is called. Note that the system name is used as the unique identifier when validating security information for a specific user ID: If two different system objects contain different system names that represent the same physical iSeries system, the user ID and password require separate validation for the two system objects. For example, this applies if the system names "SYS1" and "SYS1.ACME.COM" represent the same iSeries system. This may result in double prompting, and the use of different default user IDs when connecting.

Get by using `cwbCO_GetSystemName`

Default:

There is no default, since this is explicitly set when the system object is created.

Description

Description of the configured connection to the iSeries system.

Set using `iSeriesNavigator`.

Retrieve using `cwbCO_GetDescription`

The description is stored with each system object, and never changed for that system object. If the description is changed using `iSeries Navigator`, system objects for that system that existed before the change was made are not changed. Only new system objects will contain the new description.

Default:

Blank. This may be overridden by policies.

User ID:

The user ID used to logon to the iSeries system.

Get by using `cwbCO_GetUserIDEx`

Set by using `cwbCO_SetUserIDEx`

Default:

The first time that you connect to the iSeries system which is named in the system object, you may be prompted:

- To specify a default user ID
- To specify that the default user ID should be the same as your Windows user ID
- That no default will be used

On subsequent connection attempts, the default user ID that is used will depend on which option you chose when prompted during the first connection attempt.

Password:

The password used to signon to the iSeries system.

Set by using `cwbCO_SetPassword`

Default:

Blank (no password set) if the user ID that is set in the system object never has signed on to the iSeries system that is named in the system object. If a previous successful signon or connection has been made to the iSeries system that is named in the system object, that password may be used for the next signon or connection attempt. The system will no longer cache a password in the iSeries Access for Windows volatile password cache if the password comes in through the `cwbCO_SetPassword()` API. Previously, this would have gone into the volatile (i.e. session) password cache.

Default user mode:

Controls behavior that is associated with the default user ID, including where to obtain it and whether to use it. If it is not set (if the value is `CWBCO_DEFAULT_USER_MODE_NOT_SET`), the user may be prompted to choose which behavior is desired at the time a signon is attempted.

Get by using `cwbCO_GetDefaultUserMode`

Set by using `cwbCO_SetDefaultUserMode`

Check for modify restriction by using `cwbCO_CanModifyDefaultUserMode`

Default:

`CWBCO_DEFAULT_USER_MODE_NOT_SET`

Note: The default may be overridden by policies.

Prompt mode:

Controls when iSeries Access for Windows will prompt the user for user ID and password. See the declaration comments for `cwbCO_SetPromptMode` for possible values and for associated behaviors.

Get by using `cwbCO_GetPromptMode`

Set by using `cwbCO_SetPromptMode`

Default:

`CWBCO_PROMPT_IF_NECESSARY`

Window handle:

The window handle of the calling application. If this is set, any prompting that iSeries Access for Windows does related to iSeries signon will use the window handle, and will be modal to the associated window. This means that the prompt never will be hidden UNDER the main application window if its handle is associated with the system object. If no window handle is set, the prompt might be hidden behind the main application window, if one exists.

Get by using `cwbCO_GetWindowHandle`

Set by using `cwbCO_SetWindowHandle`

Default:

NULL (not set)

Validate mode:

Specifies, when validating user ID and password, whether communication with the iSeries system to perform this validation actually occurs. See the declaration comments for `cwbCO_SetValidateMode` and `cwbCO_GetValidateMode` for possible values and for associated behaviors.

Get by using `cwbCO_GetValidateMode`

Set by using `cwbCO_SetValidateMode`

Default:

`CWBCO_VALIDATE_IF_NECESSARY`

Use Secure Sockets:

Specifies whether iSeries Access for Windows will use secure sockets to authenticate the server (iSeries system) and to encrypt data that is sent and received. There are some cases where secure sockets cannot be used (for example, when the software support for Secure Sockets has not been installed on the PC). Accordingly, an application or user request for secure sockets use may fail, either at the time the `cwbCO_UseSecureSockets` API is called, or at connect time. If no such failure occurs, then secure sockets is being used, and `cwbCO_IsSecureSockets` will return `CWB_TRUE`.

Get by using `cwbCO_IsSecureSockets`

Set by using `cwbCO_UseSecureSockets`

Check for modify restriction by using `cwbCO_CanModifyUseSecureSockets`

Default:

Whatever has been configured for this iSeries system in the System List will be used. If no configuration for this iSeries system exists, or if the configuration specifies to use the iSeries Access default, then secure sockets will not be used (`CWB_FALSE`).

Note: The default may be overridden by policies.

Port lookup mode:

Specifies how to retrieve the remote port for an iSeries host service. It specifies whether to look it up locally (on the PC), on the iSeries system, or to simply use the default ("standard") port for the specified service. If local lookup is selected, the standard TCP/IP method of lookup in the `SERVICES` file on the PC is used. If server lookup is specified, a connection to the iSeries system server mapper is made to retrieve the port number by lookup from the iSeries system service table. If either the local or server lookup method fails, then connecting to the service will fail. For more information and for possible values, see the API declaration for `cwbCO_SetPortLookupMode`.

Get by using `cwbCO_GetPortLookupMode`

Set by using `cwbCO_SetPortLookupMode`

Check for modify restriction by using `cwbCO_CanModifyPortLookupMode`

Default:

Whatever has been configured for this iSeries system in the System List will be used. If no configuration for this iSeries system exists, the default is `CWBCO_PORT_LOOKUP_SERVER`.

Note: The default may be overridden by policies.

Persistence mode:

Specifies whether the iSeries system named in this system object may be added to the System

List (if not already in the list) once a successful call to `cwbCO_Connect` has completed. See `cwbCO_SetPersistenceMode` for more information and for possible values.

Get by using `cwbCO_GetPersistenceMode`

Set by using `cwbCO_SetPersistenceMode`

Check for modify restriction by using `cwbCO_CanModifyPersistenceMode`

Default:

`CWBCO_MAY_MAKE_PERSISTENT`

Note: The default may be overridden by policies.

Connect timeout

Specifies how long iSeries Access for Windows will wait for a connection attempt to complete. This setting does not affect how long the TCP/IP communications stack will wait before giving up. The TCP/IP communications stack might timeout before the iSeries Access connection timeout has expired. See `cwbCO_SetConnectTimeout` for more information and possible values. This value may be changed for a system object at any time.

get using `cwbCO_GetConnectTimeout`

set using `cwbCO_SetConnectTimeout`

Default:

`CWBCO_CONNECT_TIMEOUT_DEFAULT`

Note: The default may be overridden by policies.

iSeries Access for Windows Communications and Security system object APIs listing

The following Communications and Security system object APIs are listed alphabetically, by function:

Function	Communications and Security system object APIs
Create and delete a system object	<code>cwbCO_CreateSystem</code> <code>cwbCO_CreateSystemLike</code> <code>cwbCO_DeleteSystem</code>
Connect to and disconnect from the iSeries system, and for related behavior	<code>cwbCO_Connect</code> <code>cwbCO_Verify</code> <code>cwbCO_Disconnect</code> <code>cwbCO_IsConnected</code> <code>cwbCO_SetPersistenceMode</code> <code>cwbCO_GetPersistenceMode</code> <code>cwbCO_SetConnectTimeout</code> <code>cwbCO_GetConnectTimeout</code>

Function	Communications and Security system object APIs
Security validation and data	cwbCO_SetUserIDEx cwbCO_GetUserIDEx cwbCO_SetPassword cwbCO_SetValidateMode cwbCO_GetValidateMode cwbCO_SetDefaultUserMode cwbCO_GetDefaultUserMode cwbCO_SetPromptMode cwbCO_GetPromptMode cwbCO_GetWindowHandle cwbCO_SetWindowHandle cwbCO_Signon cwbCO_HasSignedOn cwbCO_VerifyUserIDPassword cwbCO_GetSignonDate cwbCO_GetPrevSignonDate cwbCO_GetPasswordExpireDate cwbCO_GetFailedSignons cwbCO_ChangePassword
Get and set other system object attributes, or determine if they can be set (if they are restricted by policies)	cwbCO_GetDescription cwbCO_GetSystemName cwbCO_UseSecureSockets cwbCO_IsSecureSockets cwbCO_SetPortLookupMode cwbCO_GetPortLookupMode cwbCO_SetIPAddressLookupMode cwbCO_GetIPAddressLookupMode cwbCO_SetIPAddress cwbCO_GetIPAddress cwbCO_CanModifyDefaultUserMode cwbCO_CanModifyIPAddressLookupMode cwbCO_CanModifyIPAddress cwbCO_CanModifyPortLookupMode cwbCO_CanModifyPersistenceMode cwbCO_CanModifyUseSecureSockets cwbCO_GetHostCCSID cwbCO_GetHostVersionEx

cwbCO_CanModifyDefaultUserMode

Purpose: Indicates whether the default user mode for the specified system object may be modified.

Syntax:

```
UINT CWB_ENTRY cwbCO_CanModifyDefaultUserMode(  
    cwbCO_SysHandle    system,  
    cwb_Boolean        *canModify );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean *canModify - output

Set to `CWB_TRUE` if this mode may be modified, otherwise set to `CWB_FALSE`.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The `canModify` pointer is NULL.

Usage: This value may not be modified if policy settings prohibit its modification, or if a successful signon or connection that is using the specified system object already has occurred. In these cases, `canModify` will be set to `CWB_FALSE`. The results returned from this API are correct only at the time of the call.

If policy settings are changed or a signon or connection is performed using this system object, the results of this API could become incorrect. This must be considered and managed, especially in a multi-threaded application.

cwbCO_CanModifyIPAddress

Purpose: Indicates whether IP Address that is used to connect may be modified for this system object.

Syntax:

```
UINT CWB_ENTRY cwbCO_CanModifyIPAddress(  
    cwbCO_SysHandle system,  
    cwb_Boolean *canModify );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean *canModify - output

Set to `CWB_TRUE` if the IP Address may be modified, otherwise set to `CWB_FALSE`.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The `canModify` pointer is NULL.

Usage: This value may not be modified if policy settings prohibit its modification, or if a successful signon or connection by using the specified system object already has occurred. In these cases, `canModify` will be set to `CWB_FALSE`. This value may not be modified if the IP Address Lookup Mode is not `CWBCO_IPADDR_LOOKUP_NEVER`, and policy settings prohibit modification of the IP Address Lookup Mode. In that case, `canModify` will be set to `CWB_FALSE`. The results returned from this API are correct only at the time of the call. If policy settings are changed or a signon or connection is performed using this system object, the results of this API could become incorrect. This must be considered and managed, especially in a multi-threaded application.

cwbCO_CanModifyIPAddressLookupMode

Purpose: Indicates whether the IP Address Lookup Mode may be modified for this system object.

Syntax:

```
UINT CWB_ENTRY cwbCO_CanModifyIPAddressLookupMode(  
    cwbCO_SysHandle    system,  
    cwb_Boolean        *canModify );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean *canModify - output

Set to `CWB_TRUE` if this mode may be modified, otherwise set to `CWB_FALSE`.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The `canModify` pointer is NULL.

Usage: This value may not be modified if policy settings prohibit its modification, or if a successful signon or connection using the specified system object already has occurred. In these cases, `canModify` will be set to `CWB_FALSE`. The results returned from this API are correct only at the time of the call.

If policy settings are changed or a signon or connection is performed using this system object, the results of this API could become incorrect. This must be considered and managed, especially in a multi-threaded application.

cwbCO_CanModifyPersistenceMode

Purpose: Indicates whether persistence mode for the specified system object may be modified.

Syntax:

```
UINT CWB_ENTRY cwbCO_CanModifyPersistenceMode(  
    cwbCO_SysHandle system,  
    cwb_Boolean *canModify );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean *canModify - output

Set to `CWB_TRUE` if this mode may be modified, otherwise set to `CWB_FALSE`.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The `canModify` pointer is NULL.

Usage: This value may not be modified if policy settings prohibit its modification, or if a successful signon or connection by using the specified system object has already occurred. In these cases, `canModify` will be set to `CWB_FALSE`. The results returned from this API are correct only at the time of the call. If policy settings are changed or a signon or connection is performed using this system object, the results of this API could become incorrect. This must be considered and managed, especially in a multi-threaded application.

cwbCO_CanModifyPortLookupMode

Purpose: Indicates whether the port lookup mode for the specified system object may be modified.

Syntax:

```
UINT CWB_ENTRY cwbCO_CanModifyPortLookupMode(  
    cwbCO_SysHandle system,  
    cwb_Boolean *canModify );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean *canModify - output

Set to `CWB_TRUE` if this mode may be modified, otherwise set to `CWB_FALSE`.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The `canModify` pointer is NULL.

Usage: This value may not be modified if policy settings prohibit its modification, or if a successful signon or connection by using the specified system object already has occurred. In these cases, `canModify` will be set to `CWB_FALSE`. The results returned from this API are correct only at the time of the call. If policy settings are changed or a signon or connection is performed using this system object, the results of this API could become incorrect. This must be considered and managed, especially in a multi-threaded application.

cwbCO_CanModifyUseSecureSockets

Purpose: Indicates whether the secure sockets use setting may be modified for this system object.

Syntax:

```
UINT CWB_ENTRY cwbCO_CanModifyUseSecureSockets(  
    cwbCO_SysHandle system,  
    cwb_Boolean *canModify );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean *canModify - output

Set to `CWB_TRUE` if the secure sockets use setting may be modified, otherwise set to `CWB_FALSE`.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The `canModify` pointer is NULL.

Usage: This value may not be modified if policy settings prohibit its modification, or if a successful signon or connection using the specified system object has already occurred. In these cases, `canModify` will be set to `CWB_FALSE`. The results returned from this API are correct only at the time of the call. If policy settings are changed or a signon or connection is performed using this system object, the results of this API could become incorrect. This must be considered and managed, especially in a multi-threaded application.

cwbCO_ChangePassword

Purpose: Changes the password of the specified user on the iSeries system from a specified old to a specified new value. This API does NOT use the user ID and password that currently are set in the given system object, nor does it change these values.

Syntax:

```
UINT CWB_ENTRY cwbCO_ChangePassword(  
    cwbCO_SysHandle    system,  
    LPCSTR             userID,  
    LPCSTR             oldPassword,  
    LPCSTR             newPassword,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

LPCSTR userID - input

A pointer to an ASCIIZ string that contains the user ID. The maximum length is `CWBCO_MAX_USER_ID + 1` characters, including the null terminator.

LPCSTR oldPassword - input

A pointer to a buffer which contains the old password. The maximum length is `CWBCO_MAX_PASSWORD + 1` bytes, including the null terminator.

LPCSTR newPassword - input

A pointer to a buffer which contains the new password. The maximum length is `CWBCO_MAX_PASSWORD + 1` bytes, including the null terminator.

cwbSV_ErrHandle errorHandler - input/output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, or if the errorHandler is invalid, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

A pointer parameter is NULL.

CWB_GENERAL_SECURITY_ERROR

A general security error occurred. The user profile does not have a password or the password validation program found an error in the password.

CWB_INVALID_PASSWORD

One or more characters in the new password is invalid or the password is too long.

CWB_INVALID_USERID

One or more characters in the user ID is invalid or the user ID is too long.

CWB_UNKNOWN_USERID

The supplied user ID is not known to this system.

CWB_WRONG_PASSWORD

Password is not correct.

CWB_USER_PROFILE_DISABLED

The user ID has been disabled.

CWB_PW_TOO_LONG

New password longer than maximum accepted length.

CWB_PW_TOO_SHORT

New password shorter than minimum accepted length.

CWB_PW_REPEAT_CHARACTER

New password contains a character used more than once.

CWB_PW_ADJACENT_DIGITS

New password has adjacent digits.

CWB_PW_CONSECUTIVE_CHARS

New password contains a character repeated consecutively.

CWB_PW_PREVIOUSLY_USED

New password was previously used.

CWB_PW_DISALLOWED_CHAR

New password uses an installation-disallowed character.

CWB_PW_NEED_NUMERIC

New password must contain at least one numeric.

CWB_PW_MATCHES_OLD

New password matches old password in one or more character positions.

CWB_PW_NOT_ALLOWED

New password exists in a dictionary of disallowed passwords.

CWB_PW_CONTAINS_USERID

New password contains user ID as part of the password.

CWB_PW_LAST_INVALID_PWD

The next invalid password will disable the user profile.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: Valid password lengths depend on the current setting of the iSeries system password level. Password levels 0 and 1 allow passwords up to 10 characters in length. Password levels 2 and 3 allow passwords up to 128 characters in length.

cwbCO_Connect

Purpose: Connect to the specified iSeries host service.

Syntax:

```
UINT CWB_ENTRY cwbCO_Connect(  
                                cwbCO_SysHandle    system,  
                                cwbCO_Service      service,  
                                cwbSV_ErrHandle    errorHandle );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system to connect to.

cwbCO_Service service - input

The service to connect to on the iSeries system. Valid values are those listed in “Defines for `cwbCO_Service`” on page 108, except for the values `CWBCO_SERVICE_ANY` and `CWBCO_SERVICE_ALL`. Only one service may be specified for this API, unlike for `cwbCO_Disconnect`, which can disconnect multiple services at once.

cwbSV_ErrHandle errorHandle - input/output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, or if the `errorHandle` is invalid, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_SERVICE_NAME_ERROR

The service identifier is not a valid value, or was a combination of values (only a single value is allowed for this API).

CWB_CONNECTION_TIMED_OUT

It took too long to find the iSeries system, so the attempt timed out.

CWB_CONNECTION_REFUSED

The iSeries system refused to accept our connection attempt.

CWB_NETWORK_IS_DOWN

A network error occurred, or TCP/IP is not configured correctly on the PC.

CWB_NETWORK_IS_UNREACHABLE

The network segment to which the iSeries system is connected currently is not reachable from the segment to which the PC is connected.

CWB_TIMED_OUT

The connect timeout value associated with the system object expired before the connection attempt completed, so we stopped waiting.

Note: Other return codes may be commonly returned as the result of a failed security validation attempt. See the list of common return codes in the comments for `cwbCO_Signon`.

Usage: If `signon` to the iSeries system has not yet occurred, the `signon` will be performed first when `cwbCO_Connect` is called. If you want the `signon` to occur at a separate time, call `cwbCO_Signon` first,

then call `cwbCO_Connect` at a later time. For more information about signon and its behavior, see comments for `cwbCO_Signon`. If the signon attempt fails, a connection to the specified service will not be established.

If the iSeries system as named in the specified system object does not exist in the System List, and the system object Persistence Mode is set appropriately, then when `cwbCO_Connect` or `cwbCO_Signon` is first successfully called, the iSeries system, as named in the system object, will be added to the System List. For more information about the Persistence Mode, see the comments for `cwbCO_SetPersistenceMode`.

If a connection to the specified service already exists, no new connection will be established, and `CWB_OK` will be returned. Each time this API is successfully called, the usage count for the connection to the specified service will be incremented.

Each time `cwbCO_Disconnect` is called for the same service, the usage count will be decremented. When the usage count reaches zero, the actual connection is ended.

Therefore, it is VERY IMPORTANT that for every call to the `cwbCO_Connect` API there is a later paired call to the `cwbCO_Disconnect` API, so that the connection can be ended at the appropriate time. The alternative is to call the `cwbCO_Disconnect` API, specifying `CWBCO_SERVICE_ALL`, which will disconnect all existing connections to ALL services madethrough the specified system object, and reset all usage counts to 0.

If the return code is `CWB_TIMED_OUT`, you may want to increase the connect timeout value for this system object, by calling `cwbCO_SetConnectTimeout`, and try connecting again. If you want iSeries Access to not give up until the TCP/IP communication stack itself does, set the connect timeout to `CWBCO_CONNECT_TIMEOUT_NONE`, and try connecting again.

cwbCO_CreateSystem

Purpose: Create a new system object and return a handle to it that can be used with subsequent calls. The system object has many attributes that can be set or retrieved. See “System object attributes” on page 45 for more information.

Syntax:

```
UINT CWB_ENTRY cwbCO_CreateSystem(  
                                LPCSTR      systemName,  
                                cwbCO_SysHandle *system);
```

Parameters:

LPCSTR systemName - input

Pointer to a buffer that contains the NULL-terminated name of the iSeries system. This can be its host name, or the iSeries system’s dotted-decimal IP address itself. It must not be zero length and must not contain blanks. If the name specified is not a valid iSeries system host name or IP address string (in the form “nnn.nnn.nnn.nnn”), any connection attempt or security validation attempt will fail.

cwbCO_SysHandle *system - output

The system object handle is returned in this parameter.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_POINTER

One of the pointer parameters is NULL.

CWB_INVALID_SYSNAME

The system name is not valid.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from creating a system object for a system not already defined in the System List.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage that is being used.

Usage: When you are done using the system object, you must call `cwbCO_DeleteSystem` to free resources the system object is using. If you want to create a system object that is like one you already have, use `cwbCO_CreateSystemLike`.

cwbCO_CreateSystemLike

Purpose: Create a new system object that is similar to a given system object. You may either provide a specific system name for the new system object, or specify NULL to use the given system object's name. All attributes of the given system object are copied into the new one, with the following exceptions:

- User ID
- Password
- System name, if a different one is specified
- IP address, when the system names are different.

See "System object attributes listing" on page 46 for a list of system object attributes.

Syntax:

```
UINT CWB_ENTRY cwbCO_CreateSystemLike(  
    cwbCO_SysHandle    systemToCopy,  
    LPCSTR             systemName  
    cwbCO_SysHandle    *system);
```

Parameters:

cwbCO_SysHandle systemToCopy - input

Handle that was returned by a previous call to either cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system. This is the object that will be "copied."

LPCSTR systemName - input

Pointer to a buffer that contains the NULL-terminated name of the iSeries system to use in the new system object. If NULL or the empty string is passed, the name from the given system object is copied into the new system object. If a system name is specified, it can be the host name, or the iSeries system's dotted-decimal IP address. If the name that is specified is not a valid iSeries system host name or IP address string (in the form "nnn.nnn.nnn.nnn"), any connection attempt or security validation attempt will fail.

cwbCO_SysHandle *newSystem - output

The system object handle of the new system object is returned in this parameter.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

A pointer that is supplied to the API is not valid.

CWB_INVALID_SYSNAME

The system name is not valid.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from creating a system object for a system not already defined in the System List.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage that is being used.

Usage: When you are done using the new system object, you must call cwbCO_DeleteSystem to free resources that the system object is using.

The state of the new system object might not be the same as that of the given system object, since user ID and password validation has not been performed yet for the new one. Also, the new system object has no connections associated with it, whereas the given system object may. Because of this, even though you might not be able to change attributes of the given system object because of its state, you might be able

to change the attributes of the new system object because of its possibly different state.

cwbCO_DeleteSystem

Purpose: Deletes the system object that is specified by its handle, and frees all resources the system object has used.

Syntax:

```
UINT CWB_ENTRY cwbCO_DeleteSystem(  
                                cwbCO_SysHandle    system);
```

Parameters:

cwbCO_SysHandle system - input

Handle that was returned by a previous call to either `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

Usage: Before the system object resources are freed, if there are any connections that were made using the specified system object, they will be ended, forcefully if necessary. To determine if there are active connections, call `cwbCO_IsConnected`. If you want to know whether disconnecting any existing connections was successful, call `cwbCO_Disconnect` explicitly before calling this API.

cwbCO_Disconnect

Purpose: Disconnect from the specified iSeries host service.

Syntax:

```
UINT CWB_ENTRY cwbCO_Disconnect(  
                                cwbCO_SysHandle    system,  
                                cwbCO_Service       service,  
                                cwbSV_ErrHandle     errorHandle );
```

Parameters:

cwbCO_SysHandle system - input

Handle that was returned by a previous call to either `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system from which to disconnect.

cwbCO_Service service - input

The service from which to disconnect on the iSeries system. Valid values are those listed at the start of this file, except for the value `CWBCO_SERVICE_ANY`. If `CWBCO_SERVICE_ALL` is specified, the connections to ALL connected services will be ended, and all connection usage counts reset back to zero.

cwbSV_ErrHandle errorHandle - input/output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, or if the `errorHandle` is invalid, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_SERVICE_NAME_ERROR

The service identifier is invalid.

CWB_NOT_CONNECTED

The single service was not connected.

Usage: This function should be called when a connection that is established by using `cwbCO_Connect` no longer is needed.

If any service specified cannot be disconnected, the return code will indicate this error. If more than one error occurs, only the first one will be returned as the API return code.

Usage Notes for individual service disconnect:

This function will cause the usage count for this system object's specified service to be decremented, and may or may not end the actual connection. For more information, read the Usage Notes for the `cwbCO_Connect` API.

Disconnecting a service that is not currently connected results in `CWB_NOT_CONNECTED`.

An individual service is gracefully disconnected.

Usage Notes for CWBCO_SERVICE_ALL:

The return code `CWB_NOT_CONNECTED` is not returned when `CWBCO_SERVICE_ALL` is specified, regardless of the number of connected services.

Requesting that all active services be disconnected may generate messages on the iSeries.

cwbCO_GetConnectTimeout

Purpose: This function gets, for the specified system object, the connection timeout value, in seconds, currently set.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetConnectTimeout(  
                                cwbCO_SysHandle    system,  
                                PULONG             timeout );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

PULONG timeout - output

Returns the timeout value, in seconds. This value will be from `CWBCO_CONNECT_TIMEOUT_MIN` to `CWBCO_CONNECT_TIMEOUT_MAX`, or will be `CWBCO_CONNECT_TIMEOUT_NONE` if no connection timeout is desired.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The timeout pointer is NULL.

Usage: None.

cwbCO_GetDefaultUserMode

Purpose: This function gets, for the specified system object, the default user mode that currently is set.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetDefaultUserMode(  
    cwbCO_SysHandle    system,  
    cwbCO_DefaultUserMode *mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_DefaultUserMode * mode - output

Returns the default user mode for this system object. See comments for `cwbCO_SetDefaultUserMode` for the list of possible values and their meanings.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The mode pointer is NULL.

Usage: None.

cwbCO_GetDescription

Purpose: This function gets the text description associated with a specified system object.

Syntax:

```
#if !( defined(CWB_ANSI_ONLY) || defined(CWB_UNICODE_ONLY) )
UINT CWB_ENTRY cwbCO_GetDescription(
    cwbCO_SysHandle  system,
    LPSTR            description,
    PULONG           length );
#endif // OEM-only selection

#if !( defined(CWB_OEM_ONLY) || defined(CWB_UNICODE_ONLY) )
UINT CWB_ENTRY cwbCO_GetDescriptionA(
    cwbCO_SysHandle  system,
    LPSTR            description,
    PULONG           length );
#endif // ANSI-only selection

#if !( defined(CWB_ANSI_ONLY) || defined(CWB_OEM_ONLY) )
UINT CWB_ENTRY cwbCO_GetDescriptionW(
    cwbCO_SysHandle  system,
    LPWSTR           description,
    PULONG           length );
#endif // UNICODE-only selection

// UNICODE/ANSI API selection
#if ( defined(CWB_UNICODE) && !( defined(CWB_OEM) || defined(CWB_ANSI) ) )
#define cwbCO_GetDescription cwbCO_GetDescriptionW
#elif ( defined(CWB_ANSI) && !( defined(CWB_OEM) || defined(CWB_UNICODE) ) )
#define cwbCO_GetDescription cwbCO_GetDescriptionA
#endif // of UNICODE/ANSI selection
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

LPSTR description - output

Pointer to a buffer that will contain the NULL-terminated description. The description will be at most `CWBCO_MAX_SYS_DESCRIPTION` characters long, not including the terminating NULL.

PULONG length - input/output

Pointer to the length of the description buffer. If the buffer is too small to hold the description, including space for the terminating NULL, the size of the buffer needed will be filled into this parameter.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

One of the pointer parameters passed in is NULL.

CWB_BUFFER_OVERFLOW

The description buffer is not large enough to hold the entire description.

cwbCO_GetFailedSignons

Purpose: Retrieves the number of unsuccessful security validation attempts since the last successful attempt.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetFailedSignons(  
                                cwbCO_SysHandle    system,  
                                PUSHORT             numberFailedAttempts);
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system.

PUSHORT numberFailedAttempts - output

A pointer to a short that will contain the number of failed logon attempts if this call is successful.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The numberFailedAttempts pointer is NULL.

CWB_INV_BEFORE_VALIDATE

The user ID and password that were set in the specified system object have not been validated yet, so this information is not available.

Usage: You successfully must have called cwbCO_VerifyUserIDPassword, cwbCO_Signon, or cwbCO_Connect before using this API. If you want to ensure that the value that is returned is recent, you either must call cwbCO_VerifyUserIDPassword explicitly, or set the Validate Mode to CWBCO_VALIDATE_ALWAYS before you call cwbCO_Signon or cwbCO_Connect.

cwbCO_GetHostCCSID

Purpose: Returns the associated CCSID of the iSeries system that is represented by the given system object that was in use when the signon to the iSeries system occurred, and that is associated with the user ID that is set in the system object.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetHostCCSID(  
                                cwbCO_SysHandle    system,  
                                PULONG             pCCSID );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

PULONG pCCSID - output

The host CCSID is copied into here if successful.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

the CCSID pointer is NULL.

CWB_DEFAULT_HOST_CCSID_USED

Host CCSID 500 is returned because this API is unable to determine the host CCSID appropriate for the user ID as set in the system object.

Usage: This API does not make or require an active connection to the host system to retrieve the associated CCSID value. However, it does depend on a prior successful connection to the host system by using the same user ID as is set in the specified system object. This is because the CCSID that is returned is the one from the specific user profile, NOT the iSeries system's default CCSID. To retrieve a host CCSID without requiring a user ID, call `cwbNL_GetHostCCSID`.

cwbCO_GetHostVersionEx

Purpose: Get the version and release level of the host.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetHostVersionEx(  
    cwbCO_SysHandle    system,  
    PULONG             version,  
    PULONG             release);
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system.

PULONG version - output

Pointer to a buffer where the version level of the system is returned.

PULONG release - output

Pointer to a buffer where the release level of the system is returned.

Return Codes: The following list shows common return values:

CWB_OK

Successful Completion.

CWB_NOT_CONNECTED

The system has never been connected to when using the currently active environment.

CWB_INVALID_POINTER

One of the pointers passed in is NULL.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate a temporary buffer.

Usage: The host version is retrieved and saved whenever a connection is made to the iSeries system. If no connection has been made yet to this iSeries system in the currently-active environment, this information will not be available, and the error code CWB_NOT_CONNECTED will be returned. If you know that a connection to the iSeries system recently was made successfully, it is likely that the version and release levels returned are current. If you want to make sure that the values are available and recently have been retrieved, call cwbCO_Signon or cwbCO_Connect for this system object first, then call cwbCO_GetHostVersionEx.

cwbCO_GetIPAddress

Purpose: This function gets, for the specified system object, the IP address of the iSeries system it represents. This is the IP address that was used to connect to the iSeries system (or was set some other way, such as by using `cwbCO_SetIPAddress`), and will be used for later connections, when using the specified system object.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetIPAddress(  
                                cwbCO_SysHandle  system,  
                                LPSTR            IPAddress,  
                                PULONG          length );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned by `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

LPSTR IPAddress - output

Pointer to a buffer that will contain the NULL-terminated IP address in dotted-decimal notation (in the form "nnn.nnn.nnn.nnn" where each "nnn" is in the range of from 0 to 255).

PULONG length - input/output

Pointer to the length of the `IPAddress` buffer. If the buffer is too small to hold the output, including room for the terminating NULL, the size of the buffer needed will be filled into this parameter and `CWB_BUFFER_OVERFLOW` will be returned.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

One of the input pointers is NULL.

CWB_BUFFER_OVERFLOW

The `IPAddress` buffer is not large enough to hold the entire `IPAddress` string.

Usage: None.

cwbCO_GetIPAddressLookupMode

Purpose: This function gets, for the specified system object, the indication of when, if ever, the iSeries system's IP address will be looked up dynamically.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetIPAddressLookupMode(  
                                cwbCO_SysHandle      system,  
                                cwbCO_IPAddressLookupMode *mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned by `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_IPAddressLookupMode * mode - output

Returns the IP address lookup mode that currently is in use. See comments for “`cwbCO_SetIPAddressLookupMode`” on page 91 for possible values and their meanings.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The mode pointer is NULL.

Usage: None.

cwbCO_GetPasswordExpireDate

Purpose: Retrieves the date and time the password will expire for the user ID that is set in the given system object on the iSeries system that it represents.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetPasswordExpireDate(  
                                cwbCO_SysHandle    system,  
                                cwb_DateTime      *expirationDateTime);
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system.

cwb_DateTime * expirationDateTime - output

A pointer to a structure that contains the date and time at which the password will expire for the current user ID, in the following format:

Bytes	Content
1 - 2	Year (Example: 1998 = 0x07CF)
3	Month (January = 0x01)
4	Day (First day = 0x01;31st day = 0x1F)
5	Hour (Midnight = 0x00;23rd hour = 0x17)
6	Minute (On the hour = 0x00; 59th minute = 0x3B)
7	Second (On the minute = 0x00; 59th second = 0x3B)
8	One-hundredth of a second (on the second = 0x00; maximum = 0x63)

Note: On a given day, the maximum time is 23 hours, 59 minutes, and 59.99 seconds. Midnight is 0 hours, 0 minutes, and 0.0 seconds on the following day.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The pointer to the cwb_DateTime structure is NULL.

CWB_INV_BEFORE_VALIDATE

The user ID and password that were set in the specified system object have not been validated yet, so this information is not available.

Usage: You successfully must have called cwbCO_VerifyUserIDPassword, cwbCO_Signon, or cwbCO_Connect before using this API. If you want to ensure that the value that is returned is recent, you either must call cwbCO_VerifyUserIDPassword explicitly, or set the Validate Mode to CWBCO_VALIDATE_ALWAYS before you call cwbCO_Signon or cwbCO_Connect.

cwbCO_GetPersistenceMode

Purpose: This function gets, for the specified system object, if the system it represents, along with its attributes, will be added to the System List (if not already in the list) once a successful signon has occurred.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetPersistenceMode(  
    cwbCO_SysHandle    system,  
    cwbCO_PersistenceMode *mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_PersistenceMode * mode - output

Returns the persistence mode. See comments for `cwbCO_SetPersistenceMode` for possible values and their meanings.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The mode pointer is NULL.

Usage: None.

cwbCO_GetPortLookupMode

Purpose: This function gets, for the specified system object, the mode or method by which host service ports are looked up when they are needed by iSeries Access for Windows to establish a service connection.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetPortLookupMode(  
    cwbCO_SysHandle    system,  
    cwbCO_PortLookupMode *mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned by `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_PortLookupMode * mode - output

Returns the host service port lookup mode. See comments for `cwbCO_SetPortLookupMode` for possible values and their meanings.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The mode pointer is NULL.

Usage: None.

cwbCO_GetPrevSignonDate

Purpose: Retrieves the date and time of the previous successful security validation.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetPrevSignonDate(  
    cwbCO_SysHandle    system,  
    cwb_DateTime      *signonDateTime);
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_DateTime * signonDateTime - output

A pointer to a structure that contains the date and time at which the previous signon occurred, in the following format:

Bytes	Content
1 - 2	Year (Example: 1998 = 0x07CF)
3	Month (January = 0x01)
4	Day (First day = 0x01;31st day = 0x1F)
5	Hour (Midnight = 0x00;23rd hour = 0x17)
6	Minute (On the hour = 0x00; 59th minute = 0x3B)
7	Second (On the minute = 0x00; 59th second = 0x3B)
8	One-hundredth of a second (on the second = 0x00; maximum = 0x63)

Note: On a given day, the maximum time is 23 hours, 59 minutes, and 59.99 seconds. Midnight is 0 hours, 0 minutes, and 0.0 seconds on the following day.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The pointer to the `cwb_DateTime` structure is NULL.

CWB_INV_BEFORE_VALIDATE

The user ID and password that were set in the specified system object have not been validated yet, so this information is not available.

Usage: You successfully must have called `cwbCO_VerifyUserIDPassword`, `cwbCO_Signon`, or `cwbCO_Connect` before using this API. If you want to ensure that the value that is returned is recent, you either must call `cwbCO_VerifyUserIDPassword` explicitly, or set the Validate Mode to `CWBCO_VALIDATE_ALWAYS` before you call `cwbCO_Signon` or `cwbCO_Connect`.

cwbCO_GetPromptMode

Purpose: This function gets, for the specified system object, the prompt mode that currently is set.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetPromptMode(  
    cwbCO_SysHandle    system,  
    cwbCO_PromptMode  *mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_PromptMode * mode - output

Returns the prompt mode. See comments for `cwbCO_SetPromptMode` for possible values and their meanings.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The mode pointer is NULL.

Usage: None.

cwbCO_GetSignonDate

Purpose: Retrieves the date and time of the current successful security validation.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetSignonDate(  
    cwbCO_SysHandle    system,  
    cwb_DateTime       *signonDateTime);
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_DateTime * signonDateTime - output

A pointer to a structure that will contain the date and time at which the current signon occurred, in the following format:

Bytes	Content
1 - 2	Year (Example: 1998 = 0x07CF)
3	Month (January = 0x01)
4	Day (First day = 0x01;31st day = 0x1F)
5	Hour (Midnight = 0x00;23rd hour = 0x17)
6	Minute (On the hour = 0x00; 59th minute = 0x3B)
7	Second (On the minute = 0x00; 59th second = 0x3B)
8	One-hundredth of a second (on the second = 0x00; maximum = 0x63)

Note: On a given day, the maximum time is 23 hours, 59 minutes, and 59.99 seconds. Midnight is 0 hours, 0 minutes, and 0.0 seconds on the following day.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The pointer to the `cwb_DateTime` structure is NULL.

CWB_INV_BEFORE_VALIDATE

The user ID and password set in the specified system object have not been validated yet, so this information is not available.

Usage: You successfully must have called `cwbCO_VerifyUserIDPassword`, `cwbCO_Signon`, or `cwbCO_Connect` before using this API. If you want to ensure that the value returned is recent, you must either call `cwbCO_VerifyUserIDPassword` explicitly, or set the Validate Mode to `CWBCO_VALIDATE_ALWAYS` before you call `cwbCO_Signon` or `cwbCO_Connect`.

cwbCO_GetSystemName

Purpose: This function gets the iSeries system name that is associated with the specified system object.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetSystemName(  
                                cwbCO_SysHandle  system,  
                                LPSTR             sysName,  
                                PULONG           length );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

LPSTR sysName - output

Pointer to a buffer that will contain the NULL-terminated system name. The name will be `CWBCO_MAX_SYS_NAME` characters long at most, not including the terminating NULL.

PULONG length - input/output

Pointer to the length of the `sysName` buffer. If the buffer is too small to hold the system name, including room for the terminating NULL, the size of the buffer needed will be filled into this parameter and `CWB_BUFFER_OVERFLOW` will be returned.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

One of the pointer parameters passed in is NULL.

CWB_BUFFER_OVERFLOW

The `sysName` buffer is not large enough to hold the entire system name.

Usage: None.

cwbCO_GetUserIDEx

Purpose: This function gets the current user ID that is associated with a specified system object. This is the user ID that is being used for connections to the iSeries server.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetUserIDEx(  
                                cwbCO_SysHandle  system,  
                                LPSTR            userID,  
                                PULONG          length );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from **cwbCO_CreateSystem** or **cwbCO_CreateSystemLike**. It identifies the iSeries system.

LPSTR userID - output

Pointer to a buffer that will contain the NULL-terminated user ID. The user ID will be at most CWBCO_MAX_USER_ID characters long.

PULONG length - input/output

Pointer to the length of the userID buffer. If the buffer is too small to hold the user ID, including space for the terminating NULL, the size of the buffer needed will be filled into this parameter.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

One of the pointer parameters passed in is NULL.

CWB_BUFFER_OVERFLOW

The userID buffer is not large enough to hold the entire user ID name.

Usage: The user ID may or may not have been validated on the iSeries system yet. To make sure it has been, call **cwbCO_Signon** or **cwbCO_Connect** before calling this API.

If no user ID has been set and a signon has not occurred for the system object, the returned user ID will be the empty string, even if a default user ID has been configured for the iSeries system.

cwbCO_GetValidateMode

Purpose: This function gets, for the specified system object, the validate mode currently set.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetValidateMode(  
    cwbCO_SysHandle    system,  
    cwbCO_ValidateMode *mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_ValidateMode * mode - output

Returns the validate mode. See comments for `cwbCO_SetValidateMode` for possible values and their meanings.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The mode pointer is NULL.

Usage: None.

cwbCO_GetWindowHandle

Purpose: This function gets, for the specified system object, the window handle, if any, that currently is associated with it.

Syntax:

```
UINT CWB_ENTRY cwbCO_GetWindowHandle(  
                                cwbCO_SysHandle    system,  
                                HWND                *windowHandle );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

HWND * pWindowHandle - output

Returns the window handle associated with the system object, or NULL if no window handle is associated with it.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The windowHandle pointer is NULL.

Usage: None.

cwbCO_HasSignedOn

Purpose: Returns an indication of whether the specified system object has "signed on" (whether the user ID and password have been validated at some point in the life of the specified system object).

Syntax:

```
UINT CWB_ENTRY cwbCO_HasSignedOn(  
    cwbCO_SysHandle    system,  
    cwb_Boolean        *signedOn );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean * signedOn - output

A pointer to a `cwb_Boolean` into which is stored the indication of "signed-on-ness." If the specified system object has signed on, it will be set to `CWB_TRUE`, otherwise it will be set to `CWB_FALSE`. (On error it will be set to `CWB_FALSE` as well.)

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The `signedOn` pointer is NULL.

Usage: A returned indication of `CWB_TRUE` does not mean that the user ID and password have been validated within a certain time period, but only that since the system object's creation, a signon has occurred. That signon may not have caused or included a connection and security validation flow to the iSeries system. This means that, even if `CWB_TRUE` is returned, the next call to the system object that requires a successful signon might connect and attempt to re-validate the user ID and password, and that validation, and hence the signon, may fail. The `signedOn` indicator reflects the results of the most-recent user ID and password validation. If user ID and password validation (signon) has occurred successfully at one time, but since then this validation has failed, `signedOn` will be set to `CWB_FALSE`.

cwbCO_IsConnected

Purpose: Find out if any, and how many, connections to the iSeries system that are using the specified system object currently exist.

Syntax:

```
UINT CWB_ENTRY cwbCO_IsConnected(  
                                cwbCO_SysHandle  system,  
                                cwbCO_Service    service,  
                                PULONG          numberOfConnections );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_Service service - input

The service to check for a connection. Any of the `cwbCO_Service` values listed in “Defines for `cwbCO_Service`” on page 108 are valid. To find out if ANY service is connected, specify `CWBCO_SERVICE_ANY`. To find out how many services are connected using this system object, specify `CWBCO_SERVICE_ALL`.

PULONG numberOfConnections - output

Used to return the number of connections active for the service(s) that are specified. If the service specified is not `CWBCO_SERVICE_ALL`, the value returned will be either 0 or 1, since there can be at most one active connection per service per system object. If `CWBCO_SERVICE_ALL` is specified, this could be from zero to the possible number of services, since one connection per service might be active.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion, all services specified are connected, or if `CWBCO_SERVICE_ANY` is specified, at least one service is connected.

CWB_NOT_CONNECTED

If a single service was specified, that service is not connected. If the value `CWBCO_SERVICE_ANY` was specified, there are NO active connections. If the value `CWBCO_SERVICE_ALL` was specified, there is at least one service that is NOT connected.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_SERVICE_NAME_ERROR

The service identifier is invalid.

CWB_INVALID_POINTER

The `numberOfConnections` parameter is NULL.

Usage: If `CWBCO_SERVICE_ALL` was specified and `CWB_NOT_CONNECTED` is returned, there may be some active connections, and the count of active connections still will be passed back. To find out how many connections through the specified system object exist, call this API and specify `CWBCO_SERVICE_ALL`. If the return code is either `CWB_OK` or `CWB_NOT_CONNECTED`, the number of connections that exist is stored in `numberOfConnections`.

cwbCO_IsSecureSockets

Purpose: This function gets (for the specified system object) whether Secure Sockets is being used (if connected), or would be attempted (if not currently connected) for a connection.

Syntax:

```
UINT CWB_ENTRY cwbCO_IsSecureSockets(  
    cwbCO_SysHandle system,  
    cwb_Boolean *inUse );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwb_Boolean * inUse - output

Returns whether iSeries Access is using, or will try to use, secure sockets for communication:

CWB_TRUE

IS in use or would be if connections active.

CWB_FALSE

NOT in use, would not try to use it.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The inUse pointer is NULL.

Usage: This flag is an indication of what iSeries Access for Windows will TRY to do for any future communications. If `CWB_TRUE` is returned, then any attempt to communicate to the iSeries system that cannot be performed using secure sockets will fail.

cwbCO_SetConnectTimeout

Purpose: This function sets, for the specified system object, the number of seconds iSeries Access for Windows will wait before giving up on a connection attempt and returning an error.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetConnectTimeout(  
                                cwbCO_SysHandle    system,  
                                ULONG               timeout );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

ULONG timeout - input

Specifies the connection timeout value, in seconds. The value must be from `CWBCO_CONNECT_TIMEOUT_MIN` to `CWBCO_CONNECT_TIMEOUT_MAX`, or if no timeout is desired, use `CWBCO_CONNECT_TIMEOUT_NONE`. If the value is below the minimum, then `CWBCO_CONNECT_TIMEOUT_MIN` will be used; if it is above the maximum, `CWBCO_CONNECT_TIMEOUT_MAX` will be used.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

Usage: If no timeout value has been suggested by policy, and none has been explicitly set using this API, the connect timeout used is `CWBCO_CONNECT_TIMEOUT_DEFAULT`.

cwbCO_SetDefaultUserMode

Purpose: This function sets, for the specified system object, the behavior with respect to any configured default user ID.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetDefaultUserMode(  
                                cwbCO_SysHandle    system,  
                                cwbCO_DefaultUserMode mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_DefaultUserMode mode - input

Specifies what will be done with the default user ID. Possible values are:

CWBCO_DEFAULT_USER_MODE_NOT_SET

No default user mode is currently in use. When this mode is active, and the Prompt Mode setting does not prohibit prompting, the user will be prompted at signon or connect time to select which of the remaining default user modes should be used from then on. The signon or connect cannot succeed until one of these other mode values is selected. Setting the Default User Mode back to this value will cause the prompt to appear the next time a default user ID is needed by iSeries Access.

CWBCO_DEFAULT_USER_USE

When no user ID has explicitly been set (by using `cwbCO_SetUserIDEx`) and a signon is to occur, use the default user ID that is configured for the iSeries system as named in the system object.

CWBCO_DEFAULT_USER_IGNORE

Specifies never to use a default user ID. When a signon takes place and no user ID has explicitly been set for this system object instance, the user will be prompted to enter a user ID if the Prompt Mode allows it (see `cwbCO_SetPromptMode` comments), and no initial value for the user ID will be filled in in the prompt.

CWBCO_DEFAULT_USER_USEWINLOGON

The user ID that is used when logging on to Windows will be used as the default if no user ID explicitly has been set for this system object (by using `cwbCO_SetUserIDEx`).

CWBCO_DEFAULT_USER_USE_KERBEROS

The kerberos principal created when logging into a Windows domain will be used as the default if no user ID has explicitly been set for this system object (using `cwbCO_SetUserIDEx`).

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_PARAMETER

The mode parameter is an invalid value.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon successfully has occurred by using the specified system object, so this setting no longer may be changed.

CWB_KERB_NOT_AVAILABLE

Kerberos security package is not available on this version of Windows.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either `cwbCO_Signon` or `cwbCO_Connect` has been called successfully for this system object. The default user mode set with this API will be ignored if a user ID has been set explicitly with the `cwbCO_SetUserIDEx` API.

Error code `CWB_KERB_NOT_AVAILABLE` will be returned if you attempt to set `CWBCO_DEFAULT_USER_USE_KERBEROS` on a Windows platform that does not support Kerberos.

cwbCO_SetIPAddress

Purpose: This function sets, for the specified system object, the IP address that will be used to connect to the iSeries system. It also changes the IP Address Lookup Mode for the system object to CWBCO_IPADDR_LOOKUP_NEVER. These changes will NOT affect any other system object that exists or is created later.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetIPAddress(  
                                cwbCO_SysHandle  system,  
                                LPCSTR           IPAddress );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system.

LPCSTR IPAddress - input

Specifies the IP address as a character string, in dotted-decimal notation ("nnn.nnn.nnn.nnn"), where each "nnn" is a decimal value ranging from 0 to 255. The IPAddress must not be longer than CWBCO_MAX_IP_ADDRESS characters, not including the terminating NULL character.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_PARAMETER

The IPAddress parameter does not contain a valid IP address.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon has successfully occurred by using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either cwbCO_Signon or cwbCO_Connect has been called successfully for this system object.

Use this API to force use of a specific IP address whenever any connection is made using the specified system object. Since the IP Address Lookup Mode is set to NEVER lookup the IP address, the address specified always will be used, unless before a connect or signon occurs, the IP Address Lookup Mode is changed by calling cwbCO_SetIPAddressLookupMode.

cwbCO_SetIPAddressLookupMode

Purpose: This function sets, for the specified system object, when iSeries Access for Windows dynamically will lookup the iSeries system's IP address when a connection is to be made. If the system name that is specified when `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike` was called is an actual IP address, this setting is ignored, because iSeries Access for Windows never needs to lookup the address.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetIPAddressLookupMode(  
                                     cwbCO_SysHandle      system,  
                                     cwbCO_IPAddressLookupMode mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_IPAddressLookupMode mode - input

Specifies when the dynamic address lookup can occur. Possible values are:

CWBCO_IPADDR_LOOKUP_ALWAYS

Every time a connection is to occur, dynamically lookup the iSeries system's IP address.

CWBCO_IPADDR_LOOKUP_1HOUR

Lookup the IP address dynamically if it has been at least one hour since the last lookup for this iSeries system.

CWBCO_IPADDR_LOOKUP_1DAY

Lookup the IP address dynamically if it has been at least one day since the last lookup for this iSeries system.

CWBCO_IPADDR_LOOKUP_1WEEK

Lookup the IP address dynamically if it has been at least one week since the last lookup for this iSeries system.

CWBCO_IPADDR_LOOKUP_NEVER

Never dynamically lookup the IP address of this iSeries system, always use the IP address that was last used for this iSeries system on this PC.

CWBCO_IPADDR_LOOKUP_AFTER_STARTUP

Lookup the IP address dynamically if Windows has been re-started since the last lookup for this iSeries system.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_PARAMETER

The mode parameter is an invalid value.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon has successfully occurred by using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either `cwbCO_Signon` or `cwbCO_Connect` has been called successfully for this system object.

Setting this to a value other than `CWB_IPADDR_LOOKUP_ALWAYS` could shorten the time to connect to the iSeries system, since the dynamic lookup may cause network traffic and take many seconds to complete. If the dynamic lookup is not performed, there is a risk that the IP address of the iSeries system will have changed and a connection will either fail or will be made to the wrong iSeries system.

cwbCO_SetPassword

Purpose: This function sets the password to associate with the specified system object. This password will be used when connecting to the iSeries server with either the cwbCO_Signon or cwbCO_Connect call, and when a user ID has been set with the cwbCO_SetUserIDEx call.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetPassword(  
                                cwbCO_SysHandle  system,  
                                LPCSTR           password );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system.

LPCSTR password - input

A pointer to a buffer that contains the NULL-terminated password. The maximum length is CWBCO_MAX_PASSWORD + 1 bytes in length, including the NULL terminator.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The password pointer is NULL.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage that is being used.

CWB_INV_AFTER_SIGNON

Signon successfully has occurred by using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either cwbCO_Signon or cwbCO_Connect has been called successfully for this system object. A password set with this API will not be used unless a corresponding user ID has been set with cwbCO_SetUserIDEx.

Valid password lengths depend on the current setting of the iSeries system password level. Password levels 0 and 1 allow passwords up to 10 characters in length. Password levels 2 and 3 allow passwords up to 128 characters in length.

cwbCO_SetPersistenceMode

Purpose: This function sets for the specified system object if the system it represents (as named in the system object), along with its attributes, may be added to the System List (if not already in the list) once a signon successfully has occurred.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetPersistenceMode(  
                                cwbCO_SysHandle    system,  
                                cwbCO_PersistenceMode mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_PersistenceMode mode - input

Specifies the persistence mode. Possible values are:

CWBCO_MAY_MAKE_PERSISTENT

If the system that is named in the specified system object is not yet in the System List, add it to the list once a successful signon has completed. This will make the system, as defined by this system object, available for selection by this AND other applications running, now or in the future, on this personal computer (until the system is deleted from this list).

CWBCO_MAY_NOT_MAKE_PERSISTENT

The system that is named in the specified system object (along with its attributes) may NOT be added to the System List.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_PARAMETER

The mode parameter is an invalid value.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon successfully has occurred by using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either `cwbCO_Signon` or `cwbCO_Connect` has been called successfully for this system object.

If the system as named in the system object already is in the System List, this setting has no effect.

cwbCO_SetPortLookupMode

Purpose: This function sets, for the specified system object, how a host server port lookup will be done.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetPortLookupMode(  
                                cwbCO_SysHandle    system,  
                                cwbCO_PortLookupMode mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned by `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_PortLookupMode mode - input

Specifies port lookup method. Possible values are:

CWBCO_PORT_LOOKUP_SERVER

Lookup of a host server port will be done by contacting the host (iSeries) server mapper each time the connection of a service is to be made when one does not yet exist. The server mapper returns the port number that is then used to connect to the desired service on the iSeries system.

CWBCO_PORT_LOOKUP_LOCAL

Lookup of a host server port will be done by lookup in the SERVICES file on the PC itself.

CWBCO_PORT_LOOKUP_STANDARD

The "standard" port—that set by default for a given host server and in use if no one has changed the services table on the iSeries system for that service—will be used to connect to the desired service.

The latter two modes eliminate the iSeries server mapper connection and its associated delay, network traffic, and load on the iSeries system.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_PARAMETER

The mode parameter is an invalid value.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon has successfully occurred by using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either `cwbCO_Signon` or `cwbCO_Connect` has been called successfully for this system object.

Use `CWBCO_PORT_LOOKUP_SERVER` to be most certain of the accuracy of the port number for a service; however, this requires an extra connection to the server mapper on the iSeries system every time a new connection to a service is to be made.

Use `CWBCO_PORT_LOOKUP_STANDARD` to achieve the best performance, although if the system administrator has changed the ports of any iSeries Access host service in the service table on that iSeries system, this mode will not work.

Use `CWBCO_PORT_LOOKUP_LOCAL` for best performance when the port for a iSeries Access host service has been changed on the iSeries system represented by the system object. For this to work, entries for each host service port must be added to a file on the PC named `SERVICES`. Each such entry must contain first the standard name of the host service (for example, "as-rmtcmd" without the quotes) followed by spaces and the port number for that service. The `SERVICES` file should be located in the Windows install directory in Windows 95/98, or in subdirectory `system32\drivers\etc` under the Windows NT install directory in Windows NT.

cwbCO_SetPromptMode

Purpose: This function sets, for the specified system object, the prompt mode, which specifies when and if the user should be prompted for user ID and password, or other information, when a signon is performed.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetPromptMode(  
    cwbCO_SysHandle    system,  
    cwbCO_PromptMode  mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_PromptMode - input

Specifies the prompt mode. Possible values are:

CWBCO_PROMPT_IF_NECESSARY

iSeries Access for Windows will prompt if either the user ID or password have not been explicitly set or cannot be retrieved from the persistent configuration for this system, the password cache (if enabled), or by some other means.

If the Default User Mode has not been set, and if for this iSeries system the user has not been prompted yet for default user ID, iSeries access for Windows will prompt for it at `cwbCO_Connect` or `cwbCO_Signon` time

CWBCO_PROMPT_ALWAYS

iSeries Access for Windows will always prompt when a signon is to occur for the specified system object, even if a successful signon using the same user ID to the same iSeries system has occurred using a different system object. Since a signon can occur only once for a system object, this means that exactly one prompt per system object will occur. Additional explicit signon calls will do nothing (including prompt). See two exceptions to using this mode in the usage notes below.

CWBCO_PROMPT_NEVER

iSeries Access for Windows never will prompt for user ID and password, or for default user ID. When this mode is used, a call to any API that requires a signon for completion (for example, `cwbCO_Signon` or `cwbCO_Connect`) will fail if either the user ID or password have not been set and cannot be programmatically retrieved (from the iSeries password cache). This mode should be used when either

- iSeries Access for Windows is running on a PC that is unattended or for some other reason cannot support end-user interaction.
- The application itself is prompting for or otherwise fetching the user ID and password, and explicitly setting them by using `cwbCO_SetUserIDEx` and `cwbCO_SetPassword`.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_PARAMETER

The mode parameter is an invalid value.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon successfully has occurred by using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either `cwbCO_Signon` or `cwbCO_Connect` has been called successfully for this system object. Setting the prompt mode to `CWBCO_PROMPT_ALWAYS` will not prompt the user in the following two cases:

- A user ID and password explicitly have been set with the `cwbCO_setUserIDEx` and `cwbCO_SetPassword` APIs.
- Use Windows logon info (`CWBCO_DEFAULT_USER_USEWINLOGON`) has been set with the `cwbCO_SetDefaultUserMode` API.

cwbCO_SetUserIDEx

Purpose: This function sets the user ID to associate with the specified system object. This user ID will be used when connecting to the iSeries server with either the `cwbCO_Signon` or `cwbCO_Connect` call.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetUserIDEx(  
                                cwbCO_SysHandle  system,  
                                LPCSTR           userID );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries server system.

LPCSTR userID - input

Pointer to a buffer that contains the NULL-terminated user ID. The user ID must not be longer than `CWBCO_MAX_USER_ID` characters, not including the terminating NULL character.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

The userID pointer is NULL.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage that is being used.

CWB_INV_AFTER_SIGNON

Signon successfully has occurred by using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either `cwbCO_Signon` or `cwbCO_Connect` has been called successfully for this system object. Setting a user ID explicitly with this API will cause any default user mode set with the `cwbCO_SetDefaultUserMode` API to be ignored.

cwbCO_SetValidateMode

Purpose: This function sets, for the specified system object, the validate mode, which affects behavior when validating the user ID and password.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetValidateMode(  
    cwbCO_SysHandle    system,  
    cwbCO_ValidateMode mode );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

cwbCO_ValidateMode mode - input

Specifies the validate mode. Possible values are:

CWBCO_VALIDATE_IF_NECESSARY

If validation of this user ID on this iSeries system has occurred from this PC within the last 24 hours, and the validation was successful, then use the results of the last validation and do not connect to validate at this time. There may be other scenarios where re-validation will occur; iSeries Access for Windows will re-validate as needed.

CWBCO_VALIDATE_ALWAYS

Communication with the iSeries system to validate user ID and password will occur every time this validation is requested or required. Setting this mode forces the validation to occur (when the system object is not signed on yet). Once a system object is signed on, this setting is ignored.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_PARAMETER

The mode parameter is an invalid value.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon has successfully occurred using the specified system object, so this setting no longer may be changed.

Usage: This API cannot be used after a successful signon has occurred for the specified system object. A signon has occurred if either `cwbCO_Signon` or `cwbCO_Connect` has been called successfully for this system object.

cwbCO_SetWindowHandle

Purpose: This function sets, for the specified system object, the window handle to use if any prompting is to be done that is associated with the system object (for example, prompting for user ID and password). When so set (to a non-NULL window handle), such a prompt would appear 'modal' to the main application window and therefore never would get hidden behind that window.

Syntax:

```
UINT CWB_ENTRY cwbCO_SetWindowHandle(  
                                cwbCO_SysHandle    system,  
                                HWND                windowHandle );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system.

HWND windowHandle - input

Specifies the window handle to associate with the system object. If NULL, no window handle is associated with the system object.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

Usage: This API may be used any time to change the window handle for the specified system object, even after a successful signon.

cwbCO_Signon

Purpose: Sign the user on to the iSeries system that is represented by the specified system object by using user ID and password.

Note: Passing an incorrect password on the cwbCO_Signon API increments the invalid signon attempts counter for the specified user. The user profile is disabled if sufficient invalid passwords are sent to the host.

Syntax:

```
UINT CWB_ENTRY cwbCO_Signon(  
                                cwbCO_SysHandle    system,  
                                cwbSV_ErrHandle    errorHandle );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system.

cwbSV_ErrHandle errorHandle - input/output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, or if the errorHandle is invalid, no messages will be retrieved.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_UNKNOWN_USERID

The supplied user ID is not known to this system.

CWB_WRONG_PASSWORD

Password is not correct.

CWB_PASSWORD_EXPIRED

Password has expired.

CWB_USER_PROFILE_DISABLED

The user ID has been disabled.

CWB_INVALID_PASSWORD

One or more characters in the password is invalid or the password is too long.

CWB_INVALID_USERID

One or more characters in the user ID is invalid or the user ID is too long.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_API_ERROR

General API failure.

CWB_USER_CANCELLED

The user cancelled the signon process.

Other return codes commonly may be returned as a result of a failed attempt to connect to the signon server. For a list of such return codes, see comments for cwbCO_Connect.

Usage: Both whether the user is prompted for user ID and password, and whether the iSeries system actually is contacted during user validation, are influenced by current system object settings, such as user ID, password, Prompt Mode, Default User Mode, and Validate Mode. See declarations for the get/set APIs of these attributes for more information. If the iSeries system as named in the specified system object does not exist in the System List, and the system object Persistence Mode is set appropriately, then when `cwbCO_Connect` or `cwbCO_Signon` first is called successfully, the iSeries system, as named in the system object, will be added to the System List.

For more information about the Persistence Mode, see the comments for `cwbCO_SetPersistenceMode`. If successful, and iSeries server password caching is enabled, the password will be stored for the resulting user ID in the PC's iSeries server password cache.

See also:

- "Differences between `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword`" on page 108
- "Similarities between `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword`" on page 108

cwbCO_UseSecureSockets

Purpose: Specify that all communication to the iServer system that uses the specified system object either must use secure sockets or must not use secure sockets.

Syntax:

```
UINT CWB_ENTRY cwbCO_UseSecureSockets(  
                                cwbCO_SysHandle system,  
                                cwb_Boolean useSecureSockets );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iServer system.

cwb_Boolean useSecureSockets - input

Specifies whether to require secure sockets use when communicating with the iServer system that the specified system object handle represents. Use the appropriate value:

CWB_TRUE

Require secure sockets use for communication

CWB_FALSE

Do not use secure sockets for communication

CWB_TIMED_OUT

The connect timeout value associated with the system object expired before the connection verification attempt completed, so we stopped waiting.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_SECURE_SOCKETS_NOTAVAIL

Secure sockets is not available. It may not be installed on the PC, prohibited for this user, or not available on the iServer system.

CWB_RESTRICTED_BY_POLICY

A policy exists that prohibits the user from changing this value.

CWB_INV_AFTER_SIGNON

Signon has successfully occurred by using the specified system object, so this setting no longer may be changed.

Usage: Even if a connection to the specified service already exists for the given system object, a new connection is attempted. The attributes of the given system object, such as whether to use secure sockets, are used for this connection attempt. It is therefore possible that connection verification may fail given the passed system object, but might succeed to the same system given a system object whose attributes are set differently. The most obvious example of this is where secure sockets use is concerned, since the non-secure-sockets version of the service may be running on the iServer system, while the secure-sockets version of the service might not be running, or vice-versa.

iSeries Access for Windows may or may not be able to detect at the time this API is called if Secure Sockets will be available for use at connect time for this iSeries system. Even if CWB_SECURE_SOCKETS_NOTAVAIL is NOT returned, it may be determined at a later time that secure sockets is not available.

cwbCO_Verify

Purpose: Verifies that a connection can be made to a specific host service on an iSeries system.

Syntax:

```
UINT CWB_ENTRY cwbCO_Verify(  
    cwbCO_SysHandle    system,  
    cwbCO_Service      service,  
    cwbSV_ErrHandle    errorHandle );
```

Parameters:

cwbCO_SysHandle system - input

Handle previously returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It identifies the iSeries system to which to verify connectability.

cwbCO_Service service - input

The service to verify connectability to on the iSeries system. Valid values are those listed in “Defines for `cwbCO_Service`” on page 108, except for the value `CWBCO_SERVICE_ANY`. To verify connectability of ALL services, specify `CWBCO_SERVICE_ALL`.

cwbSV_ErrHandle errorHandle - input/output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, or if the `errorHandle` is invalid, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_SERVICE_NAME_ERROR

The service identifier is invalid.

CWB_COMMUNICATIONS_ERROR

An error occurred attempting to verify a connection to the service.

Usage: This API does not require user ID and password to be set, nor will it cause a signon to occur, thus it will never prompt for this information. It does not change the state of the system object in any way.

If a connection to any specified service already exists, no new connection will be established, and connectability will be considered verified for that service.

If `CWBCO_SERVICE_ALL` is specified for verification, the return code will be `CWB_OK` only if ALL services can be connected to. If any one verification attempt fails, the return code will be that from the first failure, although verification of the other services still will be attempted.

Since this API does not establish a usable connection, it automatically will disconnect when the verification is complete; therefore, do NOT call `cwbCO_Disconnect` to end the connection.

cwbCO_VerifyUserIDPassword

Purpose: This function verifies the correctness of the user ID and password passed in, on the iSeries system that the specified system object represents. If the user ID and password are correct, it also retrieves data related to signon attempts and password expiration.

Note: Passing an incorrect password on the cwbCO_VerifyUserIDPassword API increments the invalid signon attempts counter for the specified user. The user profile is disabled if sufficient invalid passwords are sent to the host.

Syntax:

```
UINT CWB_ENTRY cwbCO_VerifyUserIDPassword(  
    cwbCO_SysHandle    system,  
    LPCSTR             userID,  
    LPCSTR             password,  
    cwbSV_ErrHandle   errorHandle );
```

Parameters:

cwbCO_SysHandle system - input

Handle that previously was returned from cwbCO_CreateSystem or cwbCO_CreateSystemLike. It identifies the iSeries system.

LPCSTR userID - input

Pointer to a buffer that contains the NULL-terminated user ID, which must not exceed CWBCO_MAX_USER_ID characters in length, not including the terminating NULL.

LPCSTR password - input

A pointer to a buffer that contains the NULL-terminated password. The maximum length is CWBCO_MAX_PASSWORD + 1 bytes in length, including the NULL terminator.

cwbSV_ErrHandle errorHandle - input/output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, or if the errorHandle is invalid, no messages will be retrieved.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

A pointer supplied to the API is not valid.

CWB_UNKNOWN_USERID

The supplied user ID is not known to this system.

CWB_WRONG_PASSWORD

Password is not correct.

CWB_PASSWORD_EXPIRED

Password has expired.

CWB_USER_PROFILE_DISABLED

The user ID has been disabled.

CWB_INVALID_PASSWORD

One or more characters in the password is invalid or the password is too long.

CWB_INVALID_USERID

One or more characters in the user ID is invalid or the user ID is too long.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate a temporary buffer.

CWB_API_ERROR

General API failure.

Usage: Valid password lengths depend on the current setting of the iSeries system password level. Password levels 0 and 1 allow passwords up to 10 characters in length. Password levels 2 and 3 allow passwords up to 128 characters in length.

See “Differences between `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword`” on page 108 and “Similarities between `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword`” on page 108.

Defines for cwBCO_Service

The following are values that define cwBCO_Service:

- CWBCO_SERVICE_CENTRAL
- CWBCO_SERVICE_NETFILE
- CWBCO_SERVICE_NETPRINT
- CWBCO_SERVICE_DATABASE
- CWBCO_SERVICE_ODBC
- CWBCO_SERVICE_DATAQUEUES
- CWBCO_SERVICE_REMOTECMD
- CWBCO_SERVICE_SECURITY
- CWBCO_SERVICE_DDM
- CWBCO_SERVICE_MAPI
- CWBCO_SERVICE_USF
- CWBCO_SERVICE_WEB_ADMIN
- CWBCO_SERVICE_TELNET
- CWBCO_SERVICE_MGMT_CENTRAL
- CWBCO_SERVICE_ANY
- CWBCO_SERVICE_ALL

Differences between cwBCO_Signon and cwBCO_VerifyUserIDPassword

Following are listed some of the significant differences between cwBCO_Signon and cwBCO_VerifyUserIDPassword:

- cwBCO_VerifyUserIDPassword requires that a user ID and password be passed-in (system object values for these will NOT be used), and will not prompt for this information. cwBCO_Signon may use prompting, depending on other system object settings, and in that case will use whatever values are supplied by the user for user ID and password in its validation attempt.
- Since cwBCO_VerifyUserIDPassword never will prompt for user ID and password, these settings in the specified system object will not be changed as a result of that call. A call to cwBCO_Signon, however, may change the user ID or password of the system object as the result of possible prompting for this information.
- cwBCO_VerifyUserIDPassword ALWAYS will result in a connection to the iSeries system being established to perform user ID and password validation, and to retrieve current values (such as date and time of last successful signon) related to signon attempts. cwBCO_Signon, however, might not connect to validate the user ID and password, but instead may use recent results of a previous validation. This is affected by recency of previous validation results as well as by the Validation Mode attribute of the given system object.
- The password will be cached in the iSeries password cache only in the case of the successful completion of cwBCO_Signon, never as the result of a call to cwBCO_VerifyUserIDPassword.
- cwBCO_VerifyUserIDPassword NEVER will set the system object state to 'signed on', whereas a successful cwBCO_Signon WILL change the state to 'signed on'. This is important because when a system object is in a 'signed on' state, most of its attributes may no longer be changed.

Similarities between cwBCO_Signon and cwBCO_VerifyUserIDPassword

Both APIs, when using a connection to validate the user ID and password, also retrieve current data related to signon attempts. This data then can be retrieved by using the following APIs:

- cwBCO_GetSignonDate
- cwBCO_GetPrevSignonDate
- cwBCO_GetPasswordExpireDate
- cwBCO_GetFailedSignons

iSeries Access for Windows Communications system list APIs listing

The following Communications system list APIs are listed alphabetically, by function:

Function	APIs
<p>Create a list of configured systems, either in the currently active environment or in a different environment. Retrieve the number of entries in the list, and each entry in succession.</p>	<p>cwbCO_CreateSysListHandle cwbCO_CreateSysListHandleEnv cwbCO_GetSysListSize cwbCO_GetNextSysName cwbCO_DeleteSysListHandle</p>
<p>Obtain information about individual systems that are configured or connected in the current process. Unless the environment name is passed as a parameter, these APIs work only with the currently active environment:</p>	<p>cwbCO_GetDefaultSysName cwbCO_GetConnectedSysName cwbCO_IsSystemConfigured cwbCO_IsSystemConfiguredEnv* cwbCO_IsSystemConnected cwbCO_GetUserID cwbCO_GetActiveConversations cwbCO_GetHostVersion</p>
<p>Obtain the names of environments that have been configured.</p>	<p>cwbCO_GetNumberOfEnvironments cwbCO_GetEnvironmentName cwbCO_GetActiveEnvironment</p>
<p>Determine if the calling application can modify environments and connection information.</p>	<p>cwbCO_CanConnectNewSystem cwbCO_CanSetActiveEnvironment cwbCO_CanModifyEnvironmentList cwbCO_CanModifySystemList cwbCO_CanModifySystemListEnv</p>

cwbCO_CanConnectNewSystem

Purpose: Indicates whether the user may connect to a system not currently configured in the System List within the active environment.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_CanConnectNewSystem();
```

Parameters: None

Return Codes: The following list shows common return values:

CWB_TRUE

Can connect to systems not already configured.

CWB_FALSE

Cannot connect to systems not already configured.

Usage: If this API returns CWB_FALSE, a call to cwbCO_CreateSystem with a system name not currently configured will fail, as will various other iSeries Access for Windows APIs that take system name as a parameter.

cwbCO_CanModifyEnvironmentList

Purpose: Indicates whether the user can create/remove/rename environments.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_CanModifyEnvironmentList();
```

Parameters:

None

Return Codes: The following list shows common return values.

CWB_TRUE

Can create/remove/rename/delete environments.

CWB_FALSE

Cannot create/remove/rename/delete environments.

Usage: This API indicates whether environments can be manipulated. To see if systems within an environment may be manipulated, use the `cwbCO_CanModifySystemList` and `cwbCO_CanModifySystemListEnv` APIs.

cwbCO_CanModifySystemList

Purpose: Indicates whether the user can add/remove/delete systems within the active environment. Note that systems "suggested" by the administrator via policies cannot be removed.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_CanModifySystemList();
```

Parameters:

None

Return Codes: The following list shows common return values:

CWB_TRUE

Can modify system list.

CWB_FALSE

Cannot modify system list.

Usage: This API indicates whether systems within the active environment can be manipulated. To see if environments can be manipulated see the `cwbCO_CanModifyEnvironmentList` API.

cwbCO_CanModifySystemListEnv

Purpose: Indicates whether the user can add/remove/delete systems within an input environment. Note that systems "suggested" by the administrator via policies cannot be removed.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_CanModifySystemListEnv(  
    char *environmentName);
```

Parameters:

char *environmentName - input

Pointer to a string that contains the desired environment name. If this pointer is NULL, or if it points to an empty string, the currently active environment is used.

Return Codes: The following list shows common return values:

CWB_TRUE

Can modify system list.

CWB_FALSE

Cannot modify system list, or an error occurred, such as having been passed a non-existent environment name.

Usage: This API indicates whether systems within an environment can be manipulated. To see if environments can be manipulated see the `cwbCO_CanModifyEnvironmentList` API.

cwbCO_CanSetActiveEnvironment

Purpose: Indicates whether the user can set an environment to be the active environment.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_CanSetActiveEnvironment();
```

Parameters:

None

Return Codes: The following list shows common return values:

CWB_TRUE

Can set the active environment.

CWB_FALSE

Cannot set the active environment.

Usage: None

cwbCO_CreateSysListHandle

Purpose: Creates a handle to a list of configured system names in the active environment.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_CreateSysListHandle(  
    cwbCO_SysListHandle *listHandle,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbCO_SysListHandle *listHandle - output

Pointer to a list handle that will be passed back on output. This handle is needed for other calls using the list.

cwbSV_ErrorHandle errorHandle - input

If the API call fails, the message object that is associated with this handle will be filled in with message text that describes the error. If this parameter is zero, no messages will be available.

Return Codes: The following list shows common return values:

CWB_OK

Successful Completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_POINTER

Pointer to the list handle is NULL.

Usage: `cwbCO_DeleteSysListHandle` must be called to free resources that are allocated with this API.

cwbCO_CreateSysListHandleEnv

Purpose: Creates a handle to list of configured system names of the specified environment.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_CreateSysListHandleEnv(  
    cwbCO_SysListHandle *listHandle,  
    cwbSV_ErrHandle     errorHandle,  
    LPCSTR              pEnvironment );
```

Parameters:

cwbCO_SysListHandle *listHandle - output

Pointer to a list handle that will be passed back on output. This handle is needed for other calls that are using the list.

cwbSV_ErrorHandle errorHandle - input

If the API call fails, the message object that is associated with this handle will be filled in with message text that describes the error. If this parameter is zero, no messages will be available.

LPCSTR pEnvironment

Pointer to a string containing the desired environment name. If pEnvironment is the NULL pointer, or points to the NULL string ("\"0"), the system list of the current active environment is returned.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_INVALID_POINTER

Pointer to the list handle is NULL.

CWB_NO_SUCH_ENVIRONMENT

The specified environment does not exist.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: cwbCO_DeleteSysListHandle must be called to free resources allocated with this API.

cwbCO_DeleteSysListHandle

Purpose: Deletes a handle to a list of configured system names. This must be called when you are finished using the system name list.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_DeleteSysListHandle(  
    cwbCO_SysListHandle listHandle);
```

Parameters:

cwbCO_SysListHandle - listHandle

A handle to the system name list to delete.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

Usage: Use this API to delete the list created with the `cwbCO_CreateSysListHandle` or `cwbCO_CreateSysListHandleEnv` API.

cwbCO_GetActiveConversations

Purpose: Get the number of active conversations of the system.

Syntax:

```
int CWB_ENTRY cwbCO_GetActiveConversations(  
                                           LPCSTR systemName);
```

Parameters:

LPCSTR systemName - input

Pointer to a buffer that contains the system name.

Return Codes: The number of active conversations, if any, is returned. If the systemName pointer is NULL, points to an empty string, the system is not currently connected, or system name contains one or more Unicode characters which cannot be converted, 0 will be returned.

Usage: This API returns the number of conversations active for the specified iSeries system within the CURRENT PROCESS ONLY. There may be other conversations active within other processes running on the PC.

cwbCO_GetActiveEnvironment

Purpose: Get the name of the environment currently active.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetActiveEnvironment(  
    char *environmentName,  
    unsigned long *bufferSize);
```

Parameters:

char *environmentName - output

Pointer to a buffer into which will be copied the name of the active environment, if the buffer that is passed is large enough to hold it. The buffer should be large enough to hold at least CWBCO_MAX_ENV_NAME + 1 characters, including the terminating NULL character.

unsigned long * bufferSize - input/output

input Size of the buffer pointed to by *environmentName.

output Size of buffer needed.

Return Codes: The following list shows common return values:

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

One or more pointer parameters are NULL.

CWB_BUFFER_OVERFLOW

Not enough room in output buffer to hold entire environment name. Use *bufferSize to determine the correct size. No error message is logged to the History Log since the caller is expected to recover from this error and continue.

CWBCO_NO_SUCH_ENVIRONMENT

No environments have been configured, so there is no active environment.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_API_ERROR

General API failure.

Usage:

cwbCO_GetConnectedSysName

Purpose: Get the name of the connected system corresponding to the index.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetConnectedSysName(  
    char          *systemName,  
    unsigned long *bufferSize,  
    unsigned long index);
```

Parameters:

char *systemName - output

Pointer to a buffer that will contain the system name. This buffer should be large enough to hold at least CWBCO_MAX_SYS_NAME + 1 characters, including the terminating NULL character.

unsigned long * bufferSize - input/output

input Size of the buffer pointed to by *systemName.

output

Size of buffer needed.

unsigned long index

Indicates which connected system to retrieve the name for. The first connected system's index is 0, the second index is 1, and so on.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

Pointer to system name or pointer to buffer size needed is NULL. Check messages in the History Log to determine which are NULL.

CWB_BUFFER_OVERFLOW

Not enough room in output buffer to hold entire system name. Use *bufferSize to determine the correct size. No error message is logged to the History Log since the caller is expected to recover from this error and continue.

CWBCO_END_OF_LIST

The end of connected system list has been reached. No system name was returned.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_API_ERROR

General API failure.

Usage: Connections for which system names can be retrieved are those within the current process only.

cwbCO_GetDefaultSysName

Purpose: Get the name of the default system in the active environment.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetDefaultSysName(  
    char          *defaultSystemName,  
    unsigned long  bufferSize,  
    unsigned long  *needed,  
    cwbSV_ErrHandle  errorHandle);
```

Parameters:

char *defaultSystemName - output

Pointer to a buffer that will contain the NULL-terminated system name. This buffer should be large enough to hold at least CWBCO_MAX_SYS_NAME + 1 characters, including the terminating NULL character.

unsigned long bufferSize - input

Size of input buffer.

unsigned long *needed - output

Number of bytes needed to hold entire system name including the terminating NULL.

cwbSV_ErrorHandle errorhandle - input

If the API call fails, the message object associated with this handle will be filled in with message text that describes the error. If this parameter is zero, no messages will be available.

Return Codes: The following list shows common return values:

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

Pointer to the system name or pointer to buffer size needed is NULL. Check messages in the History Log to determine which are NULL.

CWB_BUFFER_OVERFLOW

Not enough room in output buffer to hold the entire system name. Use *needed to determine the correct size. No error message is logged to the History Log since the caller is expected to recover from this error and continue.

CWBCO_DEFAULT_SYSTEM_NOT_DEFINED

The setting for the default system has not been defined in the active environment.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_API_ERROR

General API failure.

Usage:

cwbCO_GetEnvironmentName

Purpose: Get the name of the environment corresponding to the index.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetEnvironmentName(  
    char          *environmentName,  
    unsigned long *bufferSize,  
    unsigned long  index);
```

Parameters:

char *environmentName - output

Pointer to a buffer that will contain the environment name. This buffer should be large enough to hold at least CWBCO_MAX_ENV_NAME + 1 characters, including the terminating NULL character.

unsigned long * bufferSize - input/output

input Size of the buffer pointed to by *environmentName.

output

Size of buffer needed, if the buffer provided was too small.

unsigned long index - input

0 corresponds to the first environment.

Return Codes: The following list shows common return values:

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

One or more pointer parameters are NULL.

CWB_BUFFER_OVERFLOW

Not enough room in output buffer to hold entire environment name. Use *bufferSize to determine the correct size. No error message is logged to the History Log since the caller is expected to recover from this error and continue.

CWBCO_END_OF_LIST

The end of the environments list has been reached. No environment name was returned.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_API_ERROR

General API failure.

Usage:

cwbCO_GetHostVersion

Purpose: Get the version and release level of the host.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetHostVersion(  
    LPCSTR      system,  
    unsigned int * version,  
    unsigned int * release );
```

Parameters:

LPCSTR systemName - input

Pointer to a buffer that contains the system name.

unsigned int * version - output

Pointer to a buffer where the version level of the system is returned.

unsigned int * release - output

Pointer to a buffer where the release level of the system is returned.

Return Codes: The following list shows common return values:

CWB_OK

Successful Completion.

CWBCO_SYSTEM_NOT_CONFIGURED

The system is not configured in the currently active environment.

CWBCO_SYSTEM_NOT_CONNECTED

The system has never been connected to when using the currently active environment.

CWB_INVALID_POINTER

One of the pointers passed is NULL.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: The host version is retrieved and saved whenever a connection is made to the system; this API does not go to the host to get it on each call. The system must have been connected previously, though not necessarily at the time the API is called. Host information can only be retrieved for systems configured in the currently active environment.

cwbCO_GetNextSysName

Purpose: Get the name of the next system from a list of systems.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetNextSysName(  
    cwbCO_SysListHandle listHandle,  
    char                *systemName,  
    unsigned long       bufferSize,  
    unsigned long       *needed);
```

Parameters:

cwbCO_SysListHandle handleList - input

Handle to a list of systems.

char *systemName - output

Pointer to a buffer that will contain the system name. This buffer should be large enough to hold at least CWBCO_MAX_SYS_NAME + 1 characters, including the terminating NULL character.

unsigned long bufferSize - input

Size of the buffer pointed to by systemName.

unsigned long *needed - output

Number of bytes needed to hold entire system name.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

Pointer to system name or pointer to buffer size needed is NULL. Check messages in the History Log to determine which are NULL.

CWB_BUFFER_OVERFLOW

Not enough room in output buffer to hold entire system name. Use *needed to determine the correct size. No error message is logged to the History Log since the caller is expected to recover from this error and continue.

CWBCO_END_OF_LIST

The end of the system list has been reached. No system name was returned.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_API_ERROR

General API failure.

Usage: If the system list passed in was created using the API cwbCO_CreateSystemListHandle, then the system returned is configured in the currently active environment, unless between these API calls the user has removed it or switched to a different environment. If cwbCO_CreateSysListHandleEnv was called to create the system list, then the system returned is configured in the environment passed to that API, unless the user has since removed it.

cwbCO_GetNumberOfEnvironments

Purpose: Get the number of iSeries Access environments that exist. This includes both the active and all non-active environments.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetNumberOfEnvironments(  
                unsigned long    *numberOfEnv);
```

Parameters:

unsigned long *numberOfEnv - output

On output this will be set to the number of environments.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

The numberOfEnv pointer parameter is NULL.

Usage: None.

cwbCO_GetSysListSize

Purpose: Gets the number of system names in the list.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetSysListSize(  
    cwbCO_SysListHandle listHandle,  
    unsigned long      *listSize);
```

Parameters:

cwbCO_SysListHandle listHandle - input

Handle of the list of systems.

unsigned long *listSize - output

On output this will be set to the number of systems in the list.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_API_HANDLE

Invalid system handle.

CWB_INVALID_POINTER

Pointer to the list size is NULL.

Usage: None.

cwbCO_GetUserID

Purpose: Get signon or default user ID of the input system as it is configured and possibly connected in the currently active environment. This API is obsolete, and has been replaced.

Syntax:

```
unsigned int CWB_ENTRY cwbCO_GetUserID(  
    LPCSTR          systemName,  
    char            *userID,  
    unsigned int    userID_Type,  
    unsigned long   *bufferSize);
```

Parameters:

LPCSTR systemName - input

Pointer to a buffer that contains the system name.

char *userID - output

Pointer to a buffer where the desired userID of the system is returned. This buffer should be large enough to hold at least CWBCO_MAX_USER_ID + 1 characters, including the terminating NULL character.

unsigned int userID_Type - input

Specify whether current user ID of connected system (CWBCO_CURRENT_USER_ID) or default user ID of configured system (CWBCO_DEFAULT_USER_ID) is to be returned.

unsigned long * bufferSize - input/output

Pointer to a value that indicates the size of the userID buffer. If the buffer is not big enough, the value needed is returned.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

One or more input pointers are invalid.

CWB_INVALID_PARAMETER

The value for userID_Type is invalid.

CWB_BUFFER_OVERFLOW

Not enough room in userID buffer to store the user ID. Use *bufferSize to determine the correct size. No error message is logged to the History Log since the caller is expected to recover from this error and continue.

CWBCO_SYSTEM_NOT_CONNECTED

The system of the "current user ID" is not connected.

CWBCO_SYSTEM_NOT_CONFIGURED

The system of the "default user ID" is not configured in the currently active environment.

CWBCO_INTERNAL_ERROR

Internal error.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: If the default user ID is specified, and none was entered when the connection was configured, CWB_OK will be returned and the user ID sent back to the caller will be the empty string, "\0". The user ID retrieved will be that of the specified system from the currently active environment.

cwbCO_IsSystemConfigured

Purpose: Check if the input system is configured in the environment currently in use.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_IsSystemConfigured(  
                                LPCSTR    systemName);
```

Parameters:

LPCSTR systemName - input

Pointer to a buffer that contains the system name.

Return Codes: The following list shows common return values:

CWB_TRUE:

System is configured.

CWB_FALSE:

System is not configured, systemName is NULL, or system name contains one or more Unicode characters that cannot be converted.

Usage: None

cwbCO_IsSystemConfiguredEnv

Purpose: Check if the input system is configured in the environment specified.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_IsSystemConfiguredEnv(  
    LPCSTR    systemName,  
    LPCSTR    pEnvironment);
```

Parameters:

LPCSTR systemName - input

Pointer to a buffer that contains the system name.

LPCSTR pEnvironment - input

Pointer to a buffer that contains the environment name. If pEnvironment is NULL, or if it points to an empty string, the environment currently in use is checked.

Return Codes: The following list shows common return values:

CWB_TRUE:

System is configured.

CWB_FALSE:

System is not configured, systemName is NULL, or system name contains one or more Unicode characters that cannot be converted.

Usage: None

cwbCO_IsSystemConnected

Purpose: Check if the input system is currently connected.

Syntax:

```
cwb_Boolean CWB_ENTRY cwbCO_IsSystemConnected(  
                                                LPCSTR systemName);
```

Parameters:

LPCSTR systemName - input

Pointer to a buffer that contains the system name.

Return Codes: The following list shows common return values.

CWB_TRUE:

System is connected.

CWB_FALSE:

System is not connected, systemName is NULL, or system name contains one or more Unicode characters that cannot be converted.

Usage: This API indicates connection status within the current process only. The system may be connected within a different process, but this has no effect on the output of this API.

Example: Using iSeries Access for Windows communications APIs

The example program below shows the use of communications APIs to retrieve and display the names of the default (managing) system, along with all the systems that are configured in the active environment.

```

/*****
*
* Module:
*   GETSYS.C
*
* Purpose:
*   This module is used to demonstrate how an application might use the
*   Communication API's.  In this example, these APIs are used to get
*   and display the list of all configured systems.  The user can then
*   select one, and that system's connection properties (the attributes
*   of the created system object) are displayed.  All Client Access
*   services are then checked for connectability, and the results displayed.
*
* Usage Notes:
*
*   Include CWBCO.H, CWBCOSYS.H, and CWBSV.H
*   Link with CWBAPI.LIB
*
* IBM grants you a nonexclusive license to use this as an example
* from which you can generate similar function tailored to your own
* specific needs.  This sample is provided in the form of source
* material which you may change and use.
* If you change the source, it is recommended that you first copy the
* source to a different directory.  This will ensure that your changes
* are preserved when the tool kit contents are changed by IBM.
*
*                               DISCLAIMER
*                               -----
*
* This sample code is provided by IBM for illustrative purposes only.
* These examples have not been thoroughly tested under all conditions.
* IBM, therefore, cannot guarantee or imply reliability,
* serviceability, or function of these programs.  All programs
* contained herein are provided to you "AS IS" without any warranties
* of any kind.  ALL WARRANTIES, INCLUDING BUT NOT LIMITED TO THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE, ARE EXPRESSLY DISCLAIMED.
*
* Your license to this sample code provides you no right or licenses to
* any IBM patents.  IBM has no obligation to defend or indemnify against
* any claim of infringement, including but not limited to: patents,
* copyright, trade secret, or intellectual property rights of any kind.
*
*
*
*                               COPYRIGHT
*                               -----
*
*       5769-XE1 (C) Copyright IBM CORP. 1996, 1998
*       All rights reserved.
*       US Government Users Restricted Rights -
*       Use, duplication or disclosure restricted
*       by GSA ADP Schedule Contract with IBM Corp.
*       Licensed Material - Property of IBM
*
*

```

```

*
*****/

#include windows.h
#include stdio.h

#include "cwbsv.h"      /* Service APIs for retrieving any FAILURE messages */
#include "cwbc0.h"      /* Comm APIs for enumerating systems configured */
#include "cwbcosys.h"   /* Comm APIs for creating and using system objects */

#define SUCCESS    (0)
#define FAILURE    (1)

/*
 * Arrays of attribute description strings, for human-readable
 * display of these values.
 */
char* valModeStr[2] = { "CWBCO_VALIDATE_IF_NECESSARY" ,
                       "CWBCO_VALIDATE_ALWAYS" } ;

char* promptModeStr[3] = { "CWBCO_PROMPT_IF_NECESSARY" ,
                           "CWBCO_PROMPT_ALWAYS" ,
                           "CWBCO_PROMPT_NEVER" } ;

char* dfltUserModeStr[4] = { "CWBCO_DEFAULT_USER_MODE_NOT_SET" ,
                              "CWBCO_DEFAULT_USER_USE" ,
                              "CWBCO_DEFAULT_USER_IGNORE" ,
                              "CWBCO_DEFAULT_USER_USEWINLOGON" } ;

char* IPALModeStr[6] = { "CWBCO_IPADDR_LOOKUP_ALWAYS" ,
                          "CWBCO_IPADDR_LOOKUP_1HOUR" ,
                          "CWBCO_IPADDR_LOOKUP_1DAY" ,
                          "CWBCO_IPADDR_LOOKUP_1WEEK" ,
                          "CWBCO_IPADDR_LOOKUP_NEVER" ,
                          "CWBCO_IPADDR_LOOKUP_AFTER_STARTUP" } ;

char* portLookupModeStr[3] = { "CWBCO_PORT_LOOKUP_SERVER" ,
                                "CWBCO_PORT_LOOKUP_LOCAL" ,
                                "CWBCO_PORT_LOOKUP_STANDARD" } ;

char* cwbBoolStr[2] = { "False", "True" } ;

/* NOTE! The corresponding service CONSTANT integers start
 * at 1, NOT at 0; that is why the dummy "FAILURE" value
 * was added at position 0.
 */
char* serviceStr[15] = { "CWBCO_SERVICE_THISISABADSERVICE!",
                          "CWBCO_SERVICE_CENTRAL" ,
                          "CWBCO_SERVICE_NETFILE" ,
                          "CWBCO_SERVICE_NETPRINT" ,
                          "CWBCO_SERVICE_DATABASE" ,
                          "CWBCO_SERVICE_ODBC" ,
                          "CWBCO_SERVICE_DATAQUEUES" ,
                          "CWBCO_SERVICE_REMOTECMD" ,
                          "CWBCO_SERVICE_SECURITY" ,
                          "CWBCO_SERVICE_DDM" ,
                          "CWBCO_SERVICE_MAPI" ,
                          "CWBCO_SERVICE_USF" ,
                          "CWBCO_SERVICE_WEB_ADMIN" ,

```

```

        "CWBCO_SERVICE_TELNET" ,
        "CWBCO_SERVICE_MGMT_CENTRAL" } ;

```

```

/*
 * Node in a singly-linked list to hold a pointer
 * to a system name. Note that the creator of an
 * instance of this node must allocate the space to
 * hold the system name himself, only a pointer is
 * supplied here.
 */
typedef struct sysListNodeStruct SYSLISTNODE, *PSYSLISTNODE;
struct sysListNodeStruct
{
    char*          sysName;
    cwbCO_SysHandle hSys;
    PSYSLISTNODE  next;
} ;

```

```

/*****
 * Add a system name to the list of configured systems we will keep around.
 *****/
UINT addSystemToList(
    char* sysName,
    SYSLISTNODE** ppSysList )
{
    SYSLISTNODE* pNewSys;
    char* pNewSysName;

    pNewSys = (SYSLISTNODE*) malloc (sizeof( SYSLISTNODE ));
    if ( pNewSys == NULL )
    {
        return FAILURE;
    }

    pNewSysName = (char*) malloc (strlen( sysName ) + 1 );
    if ( pNewSysName == NULL )
    {
        free (pNewSys);
        return FAILURE;
    }

    strcpy( pNewSysName, sysName );
    pNewSys->sysName = pNewSysName;
    pNewSys->hSys = 0;          /* delay creating sys object until needed */
    pNewSys->next = *ppSysList;
    *ppSysList = pNewSys;

    return SUCCESS;
}

```

```

/*****
 * Clear the list of system names and clean up used storage.
 *****/

```

```

void clearList( SYSLISTNODE* pSysList )
{
    PPSYSLISTNODE pCur, pNext;

    pCur = pSysList;

    while ( pCur != NULL )
    {
        pNext = pCur->next;
        free (pCur->sysName);
        free (pCur);
        pCur = pNext;
    }
}

/*****
 * Retrieve and display Client Access FAILURE messages.
 *****/
void reportCAErrors( cwbSV_ErrHandle hErrs )
{
    ULONG msgCount;
    UINT apiRC;
    UINT i;
    char msgText[ 200 ];          /* 200 is big enuf to hold most msgs */
    ULONG bufLen = sizeof( msgText ); /* holds size of msgText buffer */
    ULONG lenNeeded;            /* to hold length of buf needed */

    apiRC = cwbSV_GetErrCount( hErrs, &msgCount );
    if ( CWB_OK != apiRC )
    {
        printf( "Failed to get message count, cwbSV_GetErrCount rc=%u\n", apiRC );
        if ( ( CWB_INVALID_POINTER == apiRC ) ||
            ( CWB_INVALID_HANDLE == apiRC ) )
        {
            printf( " --> likely a programming FAILURE!\n");
        }
        return;
    }

    bufLen = sizeof( msgText );
    for ( i=1; i<=msgCount; i++ )
    {
        apiRC = cwbSV_GetErrTextIndexed(hErrs, i, msgText, bufLen, &lenNeeded);
        if ( ( CWB_OK == apiRC ) ||
            ( CWB_BUFFER_OVERFLOW == apiRC ) ) /* if truncated, that's ok */
        {
            printf( "CA FAILURE #%u: %s\n", i, msgText );
        }
        else
        {
            printf( "CA FAILURE #%u unavailable, cwbSV_GetErrTextIndexed rc=%u\n",
                i, apiRC );
        }
    }
}

/*****
 * Build the list of systems as it is currently configured in Client
 *****/

```

```

* Access.
*****/
UINT buildSysList(
    SYSLISTNODE** ppSysList )
{
    cwbSV_ErrHandle    hErrs;
    cwbCO_SysListHandle hList;
    char               sysName[ CWBCO_MAX_SYS_NAME + 1 ];
    ULONG              bufSize = sizeof( sysName );
    ULONG              needed;
    UINT               apiRC;
    UINT               myRC = SUCCESS;
    UINT               rc = SUCCESS;

    /* Create a FAILURE handle so that, in case of FAILURE, we can
     * retrieve and display the messages (if any) associated with
     * the failure.
     */
    apiRC = cwbSV_CreateErrHandle( &hErrs );
    if ( CWB_OK != apiRC )
    {
        /* Failed to create a FAILURE handle, use NULL instead.
         * This means we'll not be able to get at FAILURE messages.
         */
        hErrs = 0;
    }

    apiRC = cwbCO_CreateSysListHandle( *hList, hErrs );
    if ( CWB_OK != apiRC )
    {
        printf( "Failure to get a handle to the system list.\n" );
        reportCAErrors( hErrs );
        myRC = FAILURE;
    }

    /* Get each successive system name and add the system to our
     * internal list for later use.
     */
    while ( ( CWB_OK == apiRC ) && ( myRC == SUCCESS ) )
    {
        apiRC = cwbCO_GetNextSysName( hList, sysName, bufSize, &needed );

        /* Note that since the sysName buffer is as large as it will
         * ever need to be, we don't check specifically for the return
         * code CWB_BUFFER_OVERFLOW. We could instead choose to use a
         * smaller buffer, and if CWB_BUFFER_OVERFLOW were returned,
         * allocate one large enough and call cwbCO_GetNextSysName
         * again.
         */
        if ( CWB_OK == apiRC )
        {
            myRC = addSystemToList( sysName, ppSysList );
            if ( myRC != SUCCESS )
            {
                printf( "Failure to add the next system name to the list.\n");
            }
        }
        else if ( CWBCO_END_OF_LIST != apiRC )
        {
            printf( "Failed to get the next system name.\n" );
        }
    }
}

```

```

        myRC = FAILURE;
    }
} /* end while (to build a list of system names) */

/*
 * Free the FAILURE handle if one was created
 */
if ( hErrs != 0 ) /* (non-NULL if it was successfully created) */
{
    apiRC = cwbsV_DeleteErrHandle( hErrs );
    if ( CWB_INVALID_HANDLE == apiRC )
    {
        printf("Failure: FAILURE handle invalid, could not delete!\n");
        myRC = FAILURE;
    }
}

return myRC;
}

/*****
 * Get a system object given an index into our list of systems.
 *****/
UINT getSystemObject(
    UINT sysNum,
    SYSLISTNODE* pSysList,
    cwbcO_SysHandle* phSys )
{
    SYSLISTNODE* pCur;
    UINT myRC, apiRC;

    pCur = pSysList;
    for ( ; sysNum > 1; sysNum-- )
    {
        /* We have come to the end of the list without finding
         * the system requested, break out of loop and set FAILURE rc.
         */
        if ( NULL == pCur )
        {
            myRC = FAILURE;
            break;
        }

        pCur = pCur->next;
    }

    /* If we're at a real system node, continue
     */
    if ( NULL != pCur )
    {
        /* We're at the node/sysname of the user's choice. If no
         * Client Access "system object" has yet been created for this
         * system, create one. Pass back the one for the selected system.
         */
        if ( 0 == pCur->hSys )
        {
            apiRC = cwbcO_CreateSystem( pCur->sysName, &(pCur->hSys) );
            if ( CWB_OK != apiRC )
            {

```

```

        printf(
            "Failed to create system object, cwbCO_CreateSystem rc = %u\n",
            apiRC );
        myRC = FAILURE;
    }
}
*phSys = pCur->hSys;
}

return myRC;
}

```

```

/*****
 * Allow the user to select a system from the list we have.
 *****/
UINT selectSystem(
    UINT* pNumSelected,
    SYSLISTNODE* pSysList,
    BOOL refreshList )
{
    UINT                myRC = SUCCESS;
    SYSLISTNODE*       pCur;
    UINT                sysNum, numSystems;
    char                choiceStr[ 20 ];

    /* If the user wants the list refreshed, clear any existing list
     * so we can rebuilt it from scratch.
     */
    if ( refreshList )
    {
        clearList( pSysList );
        pSysList = NULL;
    }

    /* If the list of system names is NULL (no list exists), build
     * the list of systems using Client Access APIs.
     */
    if ( NULL == pSysList )
    {
        myRC = buildSysList( &pSysList );
        if ( SUCCESS != myRC )
        {
            *pNumSelected = 0;
            printf( "Failed to build sys list, cannot select a system.\n" );
        }
    }

    if ( SUCCESS == myRC )
    {
        printf( "----- \n" );
        printf( "The list of systems configured is as follows:\n" );
        printf( "----- \n" );
        for ( sysNum = 1, pCur = pSysList;
              pCur != NULL;
              sysNum++, pCur = pCur->next )
        {
            printf( "  %u) %s\n", sysNum, pCur->sysName );
        }
    }
}

```



```

    numSystems = sysNum - 1;

    printf( "Enter the number of the system of your choice:\n");
    gets( choiceStr );
    *pNumSelected = atoi( choiceStr );

    if ( *pNumSelected > numSystems )
    {
        printf( "Invalid selection, there are only %u systems configured.\n");
        *pNumSelected = 0;
        myRC = FAILURE;
    }
}

return myRC;
}

```

```

/*****
 * Display a single attribute and its value, or a failing return code
 * if one occurred when trying to look it up.
 *****/
void dspAttr(
    char* label,
    char* attrVal,
    UINT lookupRC,
    BOOL* pCanBeModified,
    UINT canBeModifiedRC )
{
    if ( CWB_OK == lookupRC )
    {
        printf( "%25s : %-30s  ", label, attrVal );
        if ( CWB_OK == canBeModifiedRC )
        {
            if ( pCanBeModified != NULL )
            {
                printf( "%s\n", cwBoolStr[ *pCanBeModified ] );
            }
            else
            {
                printf( "(N/A)\n" );
            }
        }
        else
        {
            printf( "(Error, rc=%u)\n", canBeModifiedRC );
        }
    }
    else
    {
        printf( "%30s : (Error, rc=%u)\n", label, lookupRC );
    }
}

```

```

/*****
 *
 * Load the host/version string into the buffer specified. The

```

```

* buffer passed in must be at least 7 bytes long! A pointer to
* the buffer itself is passed back so that the output from this
* function can be used directly as a parameter.
*
*****/
char* hostVerModeDescr(
    ULONG ver,
    ULONG rel,
    char* verRelBuf )
{
    char* nextChar = verRelBuf;

    if ( verRelBuf != NULL )
    {
        *nextChar++ = 'v';
        if ( ver < 10 )
        {
            *nextChar++ = '0' + (char)ver;
        }
        else
        {
            *nextChar++ = '?';
            *nextChar++ = '?';
        }

        *nextChar++ = 'r';
        if ( rel < 10 )
        {
            *nextChar++ = '0' + (char)rel;
        }
        else
        {
            *nextChar++ = '?';
            *nextChar++ = '?';
        }

        *nextChar = '\0';
    }

    return verRelBuf;
}

/*****
* Display all attributes of the system whose index in the passed list
* is passed in.
*****/
void dspSysAttrs(
    SYSLISTNODE* pSysList,
    UINT sysNum )
{
    cwbCO_SysHandle hSys;
    UINT rc;
    char sysName[ CWBCO_MAX_SYS_NAME + 1 ];
    char IPAddr[ CWBCO_MAX_IP_ADDRESS + 1 ];
    ULONG bufLen, IPAddrLen;
    ULONG IPAddrBufLen;
    UINT apiRC, apiRC2;
    cwbCO_ValidateMode          valMode;

```

```

cwbCO_DefaultUserMode      dfltUserMode;
cwbCO_PromptMode           promptMode;
cwbCO_PortLookupMode       portLookupMode;
cwbCO_IPAddressLookupMode  IPALMode;
ULONG ver, rel;
char verRelBuf[ 10 ];
ULONG verRelBufLen;
cwb_Boolean isSecSoc;
cwb_Boolean canModify;

IPAddrBufLen = sizeof( IPAddr );
verRelBufLen = sizeof( verRelBuf );

rc = getSystemObject( sysNum, pSysList, &hSys );
if ( rc == FAILURE )
{
    printf( "Failed to get system object for selected system.\n");
    return;
}

printf("\n\n");
printf("-----\n");
printf("          S y s t e m   A t t r i b u t e s          \n");
printf("-----\n");
printf("\n");
printf( "%25s : %-30s   %s\n", "Attribute", "Value", "Modifiable" );
printf( "%25s : %-30s   %s\n", "-----", "-----", "-----" );
printf("\n");

apiRC = cwbCO_GetSystemName( hSys, sysName, &bufLen );
dspAttr( "System Name", sysName, apiRC, NULL, 0 );

apiRC = cwbCO_GetIPAddress( hSys, IPAddr, &IPAddrLen );
dspAttr( "IP Address", IPAddr, apiRC, NULL, 0 );

apiRC = cwbCO_GetHostVersionEx( hSys, &ver, &rel );
dspAttr( "Host Version/Release",
        hostVerModeDescr( ver, rel, verRelBuf ), apiRC, NULL, 0 );

apiRC = cwbCO_IsSecureSockets( hSys, &isSecSoc );
apiRC2 = cwbCO_CanModifyUseSecureSockets( hSys, &canModify );
dspAttr( "Secure Sockets In Use", cwbBoolStr[ isSecSoc ],
        apiRC, &canModify, apiRC2 );

apiRC = cwbCO_GetValidateMode( hSys, &valMode );
canModify = CWB_TRUE;
dspAttr( "Validate Mode", valModeStr[ valMode ], apiRC,
        &canModify, 0 );

apiRC = cwbCO_GetDefaultUserMode( hSys, &dfltUserMode );
apiRC2 = cwbCO_CanModifyDefaultUserMode( hSys, &canModify );
dspAttr( "Default User Mode", dfltUserModeStr[ dfltUserMode ], apiRC,
        &canModify, apiRC2 );

apiRC = cwbCO_GetPromptMode( hSys, &promptMode );
canModify = CWB_TRUE;
dspAttr( "Prompt Mode", promptModeStr[ promptMode ], apiRC,
        &canModify, 0 );

apiRC = cwbCO_GetPortLookupMode( hSys, &prtLookupMode );

```

```

apiRC2 = cwbcO_CanModifyPortLookupMode( hSys, &canModify );
dspAttr( "Port Lookup Mode", portLookupModeStr[ portLookupMode ], apiRC,
        &canModify, apiRC2 );

apiRC = cwbcO_GetIPAddressLookupMode( hSys, &IPALMode );
apiRC2 = cwbcO_CanModifyIPAddressLookupMode( hSys, &canModify );
dspAttr( "IP Address Lookup Mode", IPALModeStr[ IPALMode ], apiRC,
        &canModify, apiRC2 );

printf("\n\n");
}

/*****
 * Display connectability to all Client Access services that are
 * possible to connect to.
 *****/
void dspConnectability(
    PSYSLISTNODE pSysList,
    UINT sysNum )
{
    UINT rc;
    UINT apiRC;
    cwbcO_Service service;
    cwbcO_SysHandle hSys;

    rc = getSystemObject( sysNum, pSysList, &hSys );
    if ( rc == FAILURE )
    {
        printf( "Failed to get system object for selected system.\n");
    }
    else
    {
        printf("\n\n");
        printf("-----\n");
        printf("          S y s t e m   S e r v i c e s   S t a t u s           \n");
        printf("-----\n");
        for ( service=(cwbcO_Service)1;
            service <= CWBCO_SERVICE_MGMT_CENTRAL;
            service++ )
        {
            apiRC = cwbcO_Verify( hSys, service, 0 ); // 0=no err handle
            printf(" Service '%s': ", serviceStr[ service ] );
            if ( apiRC == CWB_OK )
            {
                printf("CONNECTABLE\n");
            }
            else
            {
                printf("CONNECT TEST FAILED, rc = %u\n", apiRC );
            }
        }
    }

    printf("\n");
}

```

```

/*****
* MAIN PROGRAM BODY
*****/
void main(void)
{
    PSYSLISTNODE pSysList = NULL;
    UINT numSelected;
    UINT rc;
    char choiceStr[10];
    UINT choice;

    rc = buildSysList( &pSysList );
    if ( SUCCESS != rc )
    {
        printf( "Failure to build the system list, exiting.\n\n");
        exit( FAILURE );
    }

    do
    {
        printf( "Select one of the following options:\n" );
        printf( "    (1) Display current system attributes\n");
        printf( "    (2) Display service connectability for a system\n");
        printf( "    (3) Refresh the list of systems\n" );
        printf( "    (9) Quit\n" );
        gets( choiceStr );
        choice = atoi( choiceStr );
        switch ( choice )
        {
            // ---- Display current system attributes -----
            case 1 :
            {
                rc = selectSystem( &numSelected, pSysList, FALSE );
                if ( SUCCESS == rc )
                {
                    dspSysAttrs( pSysList, numSelected );
                }

                break;
            }

            // ---- Display service connectability for a system -----
            case 2 :
            {
                rc = selectSystem( &numSelected, pSysList, FALSE );
                if ( SUCCESS == rc )
                {
                    dspConnectability( pSysList, numSelected );
                }

                break;
            }

            // ---- Refresh the list of systems -----
            case 3 :
            {
                clearList( pSysList );
                pSysList = NULL;
                rc = buildSysList( &pSysList );
                break;
            }
        }
    }
}

```

```

    }

    // ---- Quit -----
    case 9 :
    {
        printf("Ending the program!\n");
        break;
    }

    default :
    {
        printf("Invalid choice. Please make a different selection.\n");
    }
}
} while ( choice != 9 );

/* Cleanup the list, we're done */
clearList( pSysList );
pSysList = NULL;

printf( "\nEnd of program.\n\n" );
}

```

iSeries Access for Windows Data Queues APIs

Use iSeries Access for Windows Data Queues application programming interfaces (APIs) to provide easy access to iSeries data queues. Data queues allow you to create client/server applications that do not require the use of communications APIs.

iSeries Access for Windows Data Queues APIs required files:

Header file	Import library	Dynamic Link Library
cwbdq.h	cwbapi.lib	cwbdq.dll

Programmer's Toolkit:

The Programmer's Toolkit provides Data Queues documentation, access to the cwbdq.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select **Data Queues** → **C/C++ APIs**.

iSeries Access for Windows Data Queues APIs topics:

- "Data queues"
- "Ordering data queue messages" on page 145
- "Working with data queues" on page 145
- "Typical use of data queues" on page 145
- **iSeries Access for Windows Data Queues APIs listing**
- "Example: Using Data Queues APIs" on page 206
- "Data Queues APIs return codes" on page 25

Related topics:

- "iSeries system name formats for ODBC Connection APIs" on page 12
- "OEM, ANSI, and Unicode considerations" on page 12

Data queues

A data queue is a system object that exists on the iSeries system.

Benefits of using data queues:

Data queues provide many benefits to PC developers and iSeries applications developers, including:

- They are a fast and efficient means of communication on the iSeries server.
- They have low system overhead and require very little setup.
- They are efficient because a single data queue can be used by a batch job to service several interactive jobs.
- The contents of a data queue message are free-format (fields are not required), providing flexibility that is not provided by other system objects.
- Access data queues through an iSeries API and through CL commands, which provides a straight-forward means of developing client/server applications.

Ordering data queue messages

There are three ways to designate the order of messages on a data queue:

LIFO Last in, first out. The last message (newest) placed on the data queue will be the first message taken off of the queue.

FIFO First in, first out. The first message (oldest) placed on the data queue will be the first message taken off of the queue.

KEYED

Each message on the data queue has a key associated with it. A message can be taken off of the queue only by requesting the key with which it is associated.

Working with data queues

Work with data queues by using iSeries CL commands or callable programming interfaces. Access to data queues is available to all iSeries applications regardless of the programming language in which the application is written.

Use the following iSeries system interfaces to work with data queues:

OS/400 commands:

CRTDTAQ

Creates a data queue and stores it in a specified library

DLTDTAQ

Deletes the specified data queue from the system

OS/400 application programming interfaces:

QSNDDTAQ

Send a message (record) to the specified data queue

QRCVDTAQ

Read a message (record) to the specified data queue

QCLRDTAQ

Clear all messages from the specified data queue

QMHQRDQD

Retrieve a data queue description

QMHRDQM

Retrieve an entry from a data queue without removing the entry

Typical use of data queues

A data queue is a powerful program-to-program interface. Programmers who are familiar with programming on the iSeries servers are accustomed to using queues. Data queues simply represent a method that is used to pass information to another program.

Because this interface does not require communications programming, use it either for synchronous or for asynchronous (disconnected) processing.

Develop host applications and PC applications by using any supported language. For example, a host application could use RPG while a PC application might use C++. The queue is there to obtain input from one side and to pass input to the other.

The following example shows how data queues might be used:

- A PC user might take telephone orders all day, and key each order into a program, while the program places each request on iSeries data queue.
- A partner program (either a PC program or an iSeries program) monitors the data queue and pulls information from queue. This partner program could be simultaneously running, or started after peak user hours.
- It may or may not return input to the initiating PC program, or it may place something on the queue for another PC or iSeries program.
- Eventually the order is filled, the customer is billed, the inventory records are updated, and information is placed on the queue for the PC application to direct a PC user to call the customer with an expected ship date.

Objects

An application that uses the data queue function uses four **objects**. Each of these objects is identified to the application through a handle. The objects are:

Queue object:

This object represents the iSeries data queue.

Attribute:

This object describes the iSeries data queue.

Data: Use these objects to write records to, and to read records from, the iSeries data queue.

Read object:

Use this object only with the asynchronous read APIs. It uniquely identifies a request to read a record from the iSeries data queue. This handle is used on subsequent calls to check if the data has been returned. See the `cwbDQ_AsyncRead` API for more information.

iSeries Access for Windows Data Queues APIs listing

In the following table, iSeries Access for Windows Data Queues APIs are listed alphabetically, and grouped according to function:

Function	iSeries Access for Windows Data Queues APIs
Creating, deleting, and opening a data queue Use these APIs in conjunction with the cwbCO_SysHandle System Object handle.	<code>cwbDQ_CreateEx</code> <code>cwbDQ_DeleteEx</code> <code>cwbDQ_OpenEx</code>
Accessing the iSeries data queue After the cwbDQ_Open API is used to create a connection to a specific data queue, the other APIs can be used to utilize it. Use the cwbDQ_Close API when the connection no longer is needed.	<code>cwbDQ_AsyncRead</code> <code>cwbDQ_Cancel</code> <code>cwbDQ_CheckData</code> <code>cwbDQ_Clear</code> <code>cwbDQ_Close</code> <code>cwbDQ_Create</code> <code>cwbDQ_Delete</code> <code>cwbDQ_GetLibName</code> <code>cwbDQ_GetQueueAttr</code> <code>cwbDQ_GetQueueName</code> <code>cwbDQ_GetSysName</code> <code>cwbDQ_Open</code> <code>cwbDQ_Peek</code> <code>cwbDQ_Read</code> <code>cwbDQ_Write</code>

Function	iSeries Access for Windows Data Queues APIs
<p>Declarations for the attributes of a data queue</p> <p>The attribute object is used when creating a data queue or when obtaining the data queue attributes.</p>	<p>cwbDQ_CreateAttr cwbDQ_DeleteAttr cwbDQ_GetAuthority cwbDQ_GetDesc cwbDQ_GetForceToStorage cwbDQ_GetKeySize cwbDQ_GetMaxRecLen cwbDQ_GetOrder cwbDQ_GetSenderID cwbDQ_SetAuthority cwbDQ_SetDesc cwbDQ_SetForceToStorage cwbDQ_SetKeySize cwbDQ_SetMaxRecLen cwbDQ_SetOrder cwbDQ_SetSenderID</p>
<p>Declarations for functions that use the data object for writing to and reading from a data queue</p>	<p>cwbDQ_CreateData cwbDQ_DeleteData cwbDQ_GetConvert cwbDQ_GetData cwbDQ_GetDataAddr cwbDQ_GetDataLen cwbDQ_GetKey cwbDQ_GetKeyLen cwbDQ_GetRetDataLen cwbDQ_GetRetKey cwbDQ_GetRetKeyLen cwbDQ_GetSearchOrder cwbDQ_GetSenderInfo cwbDQ_SetConvert cwbDQ_SetData cwbDQ_SetDataAddr cwbDQ_SetKey cwbDQ_SetSearchOrder</p>

cwbDQ_AsyncRead

Purpose: Read a record from the iSeries data queue object that is identified by the specified handle. The **AsyncRead** will return control to the caller immediately. This call is used in conjunction with the **CheckData** API. When a record is read from a data queue, it is removed from the data queue. If the data queue is empty for more than the specified wait time, the read is aborted, and the **CheckData** API returns a value of **CWBDQ_TIMED_OUT**. You may specify a wait time from 0 to 99,999 (in seconds) or forever (-1). A wait time of zero causes the **CheckData** API to return a value of **CWBDQ_TIMED_OUT** on its initial call if there is no data in the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_AsyncRead(  
    cwbDQ_QueueHandle queueHandle,  
    cwbDQ_Data data,  
    signed long waitTime,  
    cwbDQ_ReadHandle *readHandle,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** function. This identifies the iSeries data queue object.

cwbDQ_Data data - input

The data object to be read from the iSeries data queue.

signed long waitTime - input

Length of time in seconds to wait for data, if the data queue is empty. A wait time of -1 indicates to wait forever.

cwbDQ_ReadHandle * readHandle - output

Pointer to where the **cwbDQ_ReadHandle** will be written. This handle will be used in subsequent calls to the **cwbDQ_CheckData** API.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_TIME

Invalid wait time.

CWBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

CWBDQ_INVALID_SEARCH

Invalid search order.

Usage: This function requires that you have previously issued the following APIs:

cwbDQ_Open or **cwbDQ_OpenEx**

cwbDQ_CreateData

cwbDQ_Cancel

Purpose: Cancel a previously issued **AsyncRead**. This will end the read on the iSeries data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Cancel(  
    cwbDQ_ReadHandle    readHandle,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDQ_ReadHandle readHandle - input

The handle that was returned by the **AsyncRead** API.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_READ_HANDLE

Invalid read handle.

Usage: This function requires that you have previously issued the following APIs:

cwbDQ_Open or **cwbDQ_OpenEx**

cwbDQ_CreateData

cwbDQ_AsyncRead

cwbDQ_CheckData

Purpose: Check if data was returned from a previously issued **AsyncRead** API. This API can be issued multiple times for a single **AsyncRead** call. It will return 0 when the data actually has been returned.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_CheckData(  
    cwbDQ_ReadHandle readHandle,  
    cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbDQ_ReadHandle readHandle - input

The handle that was returned by the **AsyncRead** API.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_READ_HANDLE

Invalid read handle.

CWBDQ_DATA_TRUNCATED

Data truncated.

CWBDQ_TIMED_OUT

Wait time expired and no data returned.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_QUEUE_DESTROYED

Queue was destroyed.

CWBDQ_NO_DATA

No data.

CWBDQ_CANNOT_CONVERT

Unable to convert data.

Usage: This function requires that you have previously issued the following APIs:

cwbDQ_Open or **cwbDQ_OpenEx**

cwbDQ_CreateData

cwbDQ_AsyncRead

If a time limit was specified on the **AsyncRead**, this API will return **CWBDQ_NO_DATA** until data is returned (return code will be **CWB_OK**), or the time limit expires (return code will be **CWBDQ_TIMED_OUT**).

cwbDQ_Clear

Purpose: Remove all messages from the iSeries data queue object that is identified by the specified handle. If the queue is keyed, messages for a particular key may be removed by specifying the key and key length. These values should be set to NULL and zero, respectively, if you want to clear all messages from the queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Clear(  
    cwbDQ_QueueHandle queueHandle,  
    unsigned char *key,  
    unsigned short keyLength,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** function. This identifies the iSeries data queue object.

unsigned char * key - input

Pointer to the key. The key may contain embedded NULLs, so it is not an ASCII string.

unsigned short keyLength - input

Length of the key in bytes.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

CWBDQ_BAD_KEY_LENGTH

Length of key is not correct.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

Usage: This function requires that you have previously issued:

cwbDQ_Open or **cwbDQ_OpenEx**

cwbDQ_Close

Purpose: End the connection with the iSeries data queue object that is identified by the specified handle. This will end the conversation with the iSeries system.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Close(  
                                cwbDQ_QueueHandle  queueHandle);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** or **cwbDQ_OpenEx** function. This identifies the iSeries data queue object.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

Usage: This function requires that you previously issued the following APIs:

cwbDQ_Open or **cwbDQ_OpenEx**

cwbDQ_Create

Purpose: Create an iSeries data queue object. After the object is created it can be opened using the **cwbDQ_Open** API. It will have the attributes that you specify in the attributes handle.

Note: This API is obsolete. Use “cwbDQ_CreateEx” on page 155.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Create(  
    char *queue,  
    char *library,  
    char *systemName,  
    cwbDQ_Attr queueAttributes,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

char * queue - input

Pointer to the data queue name contained in an ASCIIZ string.

char * library - input

Pointer to the library name contained in an ASCIIZ string. If this pointer is NULL then the current library will be used (set library to “*CURLIB”).

char * systemName - input

Pointer to the system name contained in an ASCIIZ string.

cwbDQ_Attr queueAttributes - input

Handle to the attributes for the data queue.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages are retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWB_SERVER_PROGRAM_NOT_FOUND

iSeries application not found.

CWB_HOST_NOT_FOUND

iSeries system inactive or does not exist.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_SECURITY_ERROR

A security error has occurred.

CWB_LICENSE_ERROR

A license error has occurred.

CWB_CONFIG_ERROR

A configuration error has occurred.

CWB_DQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

CWBDQ_BAD_QUEUE_NAME

Queue name is incorrect.

CWBDQ_BAD_LIBRARY_NAME

Library name is incorrect.

CWBDQ_BAD_SYSTEM_NAME

System name is incorrect.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_USER_EXIT_ERROR

Error in user exit program.

CWBDQ_LIBRARY_NOT_FOUND

Library not found on system.

CWBDQ_NO_AUTHORITY

No authority to library.

CWBDQ_QUEUE_EXISTS

Queue already exists.

CWBDQ_QUEUE_SYNTAX

Queue syntax is incorrect.

CWBDQ_LIBRARY_SYNTAX

Library syntax is incorrect.

Usage: This function requires that you have previously issued the following APIs:

cwbDQ_CreateAttr

cwbDQ_SetMaxRecLen

cwbDQ_CreateEx

Purpose: Create an iSeries data queue object. After the object is created it can be opened using the **cwbDQ_OpenEx** API. It will have the attributes that you specify in the attributes handle.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_CreateEx(  
    cwbCO_SysHandle sysHandle,  
    const char *queue,  
    const char *library,  
    cwbDQ_Attr queueAttributes,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbCO_SysHandle sysHandle - input

Handle to a system object

const char * queue - input

Pointer to the data queue name contained in an ASCIIZ string.

const char * library - input

Pointer to the library name contained in an ASCIIZ string. If this pointer is NULL then the current library will be used (set library to "*CURLIB").

cwbDQ_Attr queueAttributes - input

Handle to the attributes for the data queue.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWB_SERVER_PROGRAM_NOT_FOUND

iSeries application not found.

CWB_HOST_NOT_FOUND

iSeries system inactive or does not exist.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_SECURITY_ERROR

A security error has occurred.

CWB_LICENSE_ERROR

A license error has occurred.

CWB_CONFIG_ERROR

A configuration error has occurred.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

CWBDQ_BAD_QUEUE_NAME

Queue name is incorrect.

CWBDQ_BAD_LIBRARY_NAME

Library name is incorrect.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_USER_EXIT_ERROR

Error in user exit program.

CWBDQ_USER_EXIT_ERROR

Error in user exit program.

CWBDQ_LIBRARY_NOT_FOUND

Library not found on system.

CWBDQ_NO_AUTHORITY

No authority to library.

CWBDQ_QUEUE_EXISTS

Queue already exists.

CWBDQ_QUEUE_SYNTAX

Queue syntax is incorrect.

CWBDQ_LIBRARY_SYNTAX

Library syntax is incorrect.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the code page being used.

CWB_API_ERROR

General API failure.

CWB_INVALID_HANDLE

Invalid system handle.

Usage: This function requires that you have previously issued the following APIs:

cwbDQ_CreateSystem

cwbDQ_CreateAttr

cwbDQ_SetMaxRecLen

cwbDQ_CreateAttr

Purpose: Create a data queue attribute object. The handle returned by this API can be used to set the specific attributes you want for a data queue prior to using it as input for **the cwbDQ_Create** or **cwbDQ_CreateEx** APIs. It also may be used to examine specific attributes of a data queue after using it as input for the **cwbDQ_GetQueueAttr** API.

Syntax:

```
cwbDQ_Attr CWB_ENTRY cwbDQ_CreateAttr(void);
```

Parameters:

None

Return Codes: The following list shows common return values.

cwbDQ_Attr — A handle to a cwbDQ_Attr object.

Use this handle to obtain and set attributes. After creation, an attribute object will have the default values of:

- Maximum Record Length - 1000
- Order - FIFO
- Authority - LIBCRTAUT
- Force to Storage - FALSE
- Sender ID - FALSE
- Key Length - 0

Usage: None

cwbDQ_CreateData

Purpose: Create the data object. This data object can be used for both reading and writing data to a data queue.

Syntax:

```
cwbDQ_Data CWB_ENTRY cwbDQ_CreateData(void);
```

Parameters:

None

Return Codes: The following list shows common return values.

cwbDQ_Data — A handle to the data object

After creation, a data object will have the default values of:

- data - NULL and length 0
- key - NULL and length 0
- sender ID info - NULL
- search order - NONE
- convert - FALSE

Usage: None

cwbDQ_Delete

Purpose: Remove all data from an iSeries data queue and delete the data queue object.

Note: This API is obsolete. Use “cwbDQ_DeleteEx” on page 161.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Delete(  
    char *queue,  
    char *library,  
    char *systemName,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

char * queue - input

Pointer to the data queue name contained in an ASCIIZ string.

char * library - input

Pointer to the library name contained in an ASCIIZ string. If this pointer is NULL then the current library will be used (set library to “*CURLIB”).

char * systemName - input

Pointer to the system name contained in an ASCIIZ string.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWB_SERVER_PROGRAM_NOT_FOUND

iSeries application not found.

CWB_HOST_NOT_FOUND

iSeries system inactive or does not exist.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_SECURITY_ERROR

A security error has occurred.

CWB_LICENSE_ERROR

A license error has occurred.

CWB_CONFIG_ERROR

A configuration error has occurred.

CWBDQ_QUEUE_NAME

Queue name is too long.

CWBDQ_LIBRARY_NAME

Library name is too long.

CWBDQ_SYSTEM_NAME

System name is too long.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_USER_EXIT_ERROR

Error in user exit program.

CWBDQ_LIBRARY_NOT_FOUND

Library not found on system.

CWBDQ_QUEUE_NOT_FOUND

Queue not found on system.

CWBDQ_NO_AUTHORITY

No authority to queue.

CWBDQ_QUEUE_SYNTAX

Queue syntax is incorrect.

CWBDQ_LIBRARY_SYNTAX

Library syntax is incorrect.

Usage: None

cwbDQ_DeleteEx

Purpose: Remove all data from an iSeries data queue and delete the data queue object.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_DeleteEx(  
    cwbCO_SysHandle    sysHandle  
    const char        *queue,  
    const char        *library,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbCO_SysHandle - input

Handle to a system object.

const char * queue - input

Pointer to the data queue name contained in an ASCIIZ string.

const char * library - input

Pointer to the library name contained in an ASCIIZ string. If this pointer is NULL then the current library will be used (set library to *"*CURLIB"*).

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWB_SERVER_PROGRAM_NOT_FOUND

iSeries application not found.

CWB_HOST_NOT_FOUND

iSeries system inactive or does not exist.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_SECURITY_ERROR

A security error has occurred.

CWB_LICENSE_ERROR

A license error has occurred.

CWB_CONFIG_ERROR

A configuration error has occurred.

CWBDQ_BAD_QUEUE_NAME

Queue name is too long.

CWBDQ_BAD_LIBRARY_NAME

Library name is too long.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_USER_EXIT_ERROR

Error in user exit program.

CWBDQ_LIBRARY_NOT_FOUND

Library not found on system.

CWBDQ_QUEUE_NOT_FOUND

Queue not found on system.

CWBDQ_NO_AUTHORITY

No authority to queue.

CWBDQ_QUEUE_SYNTAX

Queue syntax is incorrect.

CWBDQ_LIBRARY_SYNTAX

Library syntax is incorrect.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the code page being used.

CWB_API_ERROR

General API failure.

CWB_INVALID_HANDLE

Invalid system handle.

Usage: This function requires that you previously have issued **cwbCO_CreateSystem**.

cwbDQ_DeleteAttr

Purpose: Delete the data queue attributes.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_DeleteAttr(  
    cwbDQ_Attr          queueAttributes);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_DeleteData

Purpose: Delete the data object.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_DeleteData(  
    cwbDQ_Data data);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetAuthority

Purpose: Get the attribute for the authority that other users will have to the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetAuthority(  
    cwbDQ_Attr          queueAttributes,  
    unsigned short      *authority);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

unsigned short * authority - output

Pointer to an unsigned short to where the authority will be written. This value will be one of the following defined types:

- CWBDQ_ALL
- CWBDQ_EXCLUDE
- CWBDQ_CHANGE
- CWBDQ_USE
- CWBDQ_LIBCRTAUT

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_GetConvert

Purpose: Get the value of the convert flag for a data handle. The convert flag determines if data sent to and received from the host is CCSID converted (for example, between ASCII and EBCDIC).

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetConvert(  
                                cwbDQ_Data    data,  
                                cwb_Boolean    *convert);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

cwb_Boolean * convert - output

Pointer to a Boolean where the convert flag will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetData

Purpose: Get the data attribute of the data object.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetData(  
                                cwbDQ_Data    data,  
                                unsigned char  *dataBuffer);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned char * data - output

Pointer to the data. The data may contain embedded NULLs, so it is not an ASCIIZ string.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetDataAddr

Purpose: Get the address of the location of the data buffer.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetDataAddr(  
    cwbDQ_Data data,  
    unsigned char **dataBuffer);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned char * * data - output

Pointer to where the buffer address will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

CWBDQ_ADDRESS_NOT_SET

Address not set with **cwbDQ_SetDataAddr**.

Usage: Use this function to retrieve the address of the location where the data is stored. The data address must be set with the **cwbDQ_SetDataAddr** API, otherwise, the return code **CWBDQ_ADDRESS_NOT_SET** will be returned.

cwbDQ_GetDataLen

Purpose: Get the data length attribute of the data object. This is the total length of the data object. To obtain the length of data that was read, use the **cwbDQ_GetRetDataLen** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetDataLen(  
                                cwbDQ_Data    data,  
                                unsigned long  *dataLength);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned long * dataLength - output

Pointer to an unsigned long where the length of the data will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetDesc

Purpose: Get the attribute for the description of the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetDesc(  
                                cwbDQ_Attr    queueAttributes,  
                                char          *description);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

char * description - output

Pointer to a 51 character buffer where the description will be written. The description is an ASCIIZ string.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_DQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_GetForceToStorage

Purpose: Get the attribute for whether records will be forced to auxiliary storage when they are enqueued.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetForceToStorage(  
    cwbDQ_Attr      queueAttributes,  
    cwb_Boolean     *forceToStorage);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

cwb_Boolean * forceToStorage - output

Pointer to a Boolean where the force-to-storage indicator will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_GetKey

Purpose: Get the key attribute of the data object, previously set by the **cwbDQ_SetKey** API. This is the key that is used for writing data to a keyed data queue. Along with the search order, this key is also used to read data from a keyed data queue. The key that is associated with the record retrieved can be obtained by calling the **cwbDQ_GetRetKey** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetKey(  
    cwbDQ_Data data,  
    unsigned char *key);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned char * key - output

Pointer to the key. The key may contain embedded NULLS, so it is not an ASCII string.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_DQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetKeyLen

Purpose: Get the key length attribute of the data object.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetKeyLen(  
    cwbDQ_Data data,  
    unsigned short *keyLength);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned short * keyLength - output

Pointer to an unsigned short where the length of the key will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_DQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetKeySize

Purpose: Get the attribute for the key size in bytes.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetKeySize(  
    cwbDQ_Attr      queueAttributes,  
    unsigned short *keySize);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

unsigned short * keySize - output

Pointer to an unsigned short where the key size will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_DQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_GetLibName

Purpose: Retrieve the library name used with the **cwbDQ_Open** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetLibName(  
    cwbDQ_QueueHandle queueHandle,  
    char *libName);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** function. This identifies the iSeries data queue object.

char * libName - output

Pointer to a buffer where the library name will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

Usage: This function requires that you have previously issued **cwbDQ_Open**.

cwbDQ_GetMaxRecLen

Purpose: Get the maximum record length for the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetMaxRecLen(  
    cwbDQ_Attr          queueAttributes,  
    unsigned long      *maxRecordLength);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a call to **cwbDQ_CreateAttr**.

unsigned long * maxRecordLength - output

Pointer to an unsigned long where the maximum record length will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_DQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbdQ_GetOrder

Purpose: Get the attribute for the queue order. If the order is CWBDQ_SEQ_LIFO, the last record written is the first record read (Last In First Out). If the order is CWBDQ_SEQ_FIFO, the first record written is the first record read (First In First Out). If the order is CWBDQ_SEQ_KEYED, the order in which records are read from the data queue depends on the value of the search order attribute of the data object and the key value specified for the **cwbdQ_SetKey** API. If multiple records contain the key that satisfies the search order, a FIFO scheme is used among those records.

Syntax:

```
unsigned int CWB_ENTRY cwbdQ_GetOrder(  
                                cwbdQ_Attr      queueAttributes,  
                                unsigned short  *order);
```

Parameters:

cwbdQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbdQ_CreateAttr**.

unsigned short * order - output

Pointer to an unsigned short where the order will be written. Possible values are:

- CWBDQ_SEQ_LIFO
- CWBDQ_SEQ_FIFO
- CWBDQ_SEQ_KEYED

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_GetQueueAttr

Purpose: Retrieve the attributes of the iSeries data queue object that is identified by the specified handle. A handle to the data queue attributes will be returned. The attributes then can be retrieved individually.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetQueueAttr(  
    cwbDQ_QueueHandle queueHandle,  
    cwbDQ_Attr         queueAttributes,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** function. This identifies the iSeries data queue object.

cwbDQ_Attr queueAttributes - input/output

The attribute object. This was the output from the **cwbDQ_CreateAttr** call. The attributes will be filled in by this function, and you should call the **cwbDQ_DeleteAttr** function to delete this object when you have retrieved the attributes from it.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

CWBBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

Usage: This function requires that you have previously issued the following APIs:

cwbDQ_Open or **cwbDQ_OpenEx**

cwbDQ_CreateAttr

cwbDQ_GetQueueName

Purpose: Retrieve the queue name used with the **cwbDQ_Open** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetQueueName(  
    cwbDQ_QueueHandle queueHandle,  
    char *queueName);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** function. This identifies the iSeries data queue object.

char * queueName - output

Pointer to a buffer where the queue name will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

Usage: This function requires that you have previously issued **cwbDQ_Open**.

cwbDQ_GetRetDataLen

Purpose: Get the length of data that was returned. The returned data length will be zero until a **cwbDQ_Read** or **cwbDQ_Peek** API is called. Then it will have the length of the data that actually was returned.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetRetDataLen(  
    cwbDQ_Data data,  
    unsigned long *retDataLength);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned long * retDataLength - output

Pointer to an unsigned long where the length of the data returned will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_DQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetRetKey

Purpose: Get the returned key of the data object. This is the key that is associated with the messages that are retrieved from a keyed data queue. If the search order is a value other than CWBDQ_EQUAL, this key may be different than the key that is used to retrieve the message.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetRetKey(  
    cwbDQ_Data data,  
    unsigned char *key);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned char * retKey - output

Pointer to the returned key. The key may contain embedded NULLs, so it is not an ASCII string.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetRetKeyLen

Purpose: Get the returned key length attribute of the data object. This is the length of the key that is returned by the **cwbDQ_GetKey** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetRetKeyLen(  
    cwbDQ_Data data,  
    unsigned short *retKeyLength);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned short * retKeyLength - output

Pointer to an unsigned short where the length of the key will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwBDQ_GetSearchOrder

Purpose: Get the search order of the open attributes. The search order is used when reading or peeking a keyed data queue to identify the relationship between the key of the record to retrieve and the key value specified on the **cwBDQ_SetKey** API. If the data queue order attribute is not CWBDQ_SEQ_KEYED, this property is ignored.

Syntax:

```
unsigned int CWB_ENTRY cwBDQ_GetSearchOrder(  
    cwBDQ_Data data,  
    unsigned short *searchOrder);
```

Parameters:

cwBDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwBDQ_CreateData**.

unsigned short * searchOrder - output

Pointer to an unsigned short where the order will be written. Possible values are:

- CWBDQ_NONE
- CWBDQ_EQUAL
- CWBDQ_NOT_EQUAL
- CWBDQ_GT_OR_EQUAL
- CWBDQ_GREATER
- CWBDQ_LT_OR_EQUAL
- CWBDQ_LESS

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetSenderId

Purpose: Get the attribute for whether information about the sender is kept with each record on the queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetSenderId(  
    cwbDQ_Attr          queueAttributes,  
    cwb_Boolean        *senderID);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes that are returned by a previous call to **cwbDQ_CreateAttr**.

cwb_Boolean * senderID - output

Pointer to a Boolean where the sender ID indicator will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_GetSenderInfo

Purpose: Get the Sender Information attribute of the open attributes. This information only is available if the **senderID** attribute of the Data Queue was set on creation.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetSenderInfo(  
    cwbDQ_Data data,  
    unsigned char *senderInfo);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned char * senderInfo - output

Pointer to a 36 character buffer where the sender information will be written. This buffer contains:

Job Name (10 bytes)

User Name (10 bytes)

Job ID (6 bytes)

User Profile (10 bytes)

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_GetSysName

Purpose: Retrieve the system name that is used with the **cwbDQ_Open** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_GetSysName(  
    cwbDQ_QueueHandle queueHandle,  
    char *systemName);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** function. This identifies the iSeries data queue object.

char *systemName - output

Pointer to a buffer where the system name will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_INVALID_QUEUE_HANDLE

Invalid queue handle.

Usage: This function requires that you previously have issued **cwbDQ_Open** or **cwbDQ_OpenEx**.

cwbDQ_Open

Purpose: Start a connection to the specified data queue. This will start a conversation with the iSeries system. If the connection is not successful, a non-zero handle will be returned.

Note: This API is obsolete. Use “cwbDQ_OpenEx” on page 189.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Open(  
    char *queue,  
    char *library,  
    char *systemName,  
    cwbDQ_QueueHandle *queueHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

char * queue - input

Pointer to the data queue name contained in an ASCIIZ string.

char * library - input

Pointer to the library name that is contained in an ASCIIZ string. If this pointer is NULL, the library list will be used (set library to “*LIBL”).

char * systemName - input

Pointer to the system name that is contained in an ASCIIZ string.

cwbDQ_QueueHandle * queueHandle - output

Pointer to a **cwbDQ_QueueHandle** where the handle will be returned. This handle should be used in all subsequent calls.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWB_SERVER_PROGRAM_NOT_FOUND

iSeries application not found.

CWB_HOST_NOT_FOUND

iSeries system inactive or does not exist.

CWB_COMM_VERSION_ERROR

Data Queues will not run with this version of communications.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_SECURITY_ERROR

A security error has occurred.

CWB_LICENSE_ERROR

A license error has occurred.

CWB_CONFIG_ERROR

A configuration error has occurred.

CWBDQ_BAD_QUEUE_NAME

Queue name is too long.

CWBDQ_BAD_LIBRARY_NAME

Library name is too long.

CWBDQ_BAD_SYSTEM_NAME

System name is too long.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_USER_EXIT_ERROR

Error in user exit program.

CWBDQ_LIBRARY_NOT_FOUND

Library not found on system.

CWBDQ_QUEUE_NOT_FOUND

Queue not found on system.

CWBDQ_NO_AUTHORITY

No authority to queue or library.

CWBDQ_DAMAGED_QUE

Queue is in unusable state.

CWBDQ_CANNOT_CONVERT

Data cannot be converted for this queue.

Usage: None

cwbDQ_OpenEx

Purpose: Start a connection to the specified data queue. This will start a conversation with the iSeries system. If the connection is not successful, a non-zero handle will be returned.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_OpenEx(  
    cwbCO_SysHandle    sysHandle  
    const char         *queue,  
    const char         *library,  
    cwbDQ_QueueHandle *queueHandle,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbCO_SysHandle sysHandle - input

Handle to a system object.

const char * queue - input

Pointer to the data queue name contained in an ASCIIZ string.

const char * library - input

Pointer to the library name that is contained in an ASCIIZ string. If this pointer is NULL, the library list will be used (set library to *"*LIBL"*).

cwbDQ_QueueHandle * queueHandle - output

Pointer to a cwbDQ_QueueHandle where the handle will be returned. This handle should be used in all subsequent calls.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWB_SERVER_PROGRAM_NOT_FOUND

iSeries application not found.

CWB_HOST_NOT_FOUND

iSeries system inactive or does not exist.

CWB_COMM_VERSION_ERROR

Data Queues will not run with this version of communications.

CWB_INVALID_POINTER

Bad or null pointer.

CWB_SECURITY_ERROR

A security error has occurred.

CWB_LICENSE_ERROR

A license error has occurred.

CWB_CONFIG_ERROR

A configuration error has occurred.

CWBDQ_BAD_QUEUE_NAME

Queue name is too long.

CWBDQ_BAD_LIBRARY_NAME

Library name is too long.

CWBDQ_BAD_SYSTEM_NAME

System name is too long.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_USER_EXIT_ERROR

Error in user exit program.

CWBDQ_LIBRARY_NOT_FOUND

Library not found on system.

CWBDQ_QUEUE_NOT_FOUND

Queue not found on system.

CWBDQ_NO_AUTHORITY

No authority to queue or library.

CWBDQ_DAMAGED_QUE

Queue is in unusable state.

CWBDQ_CANNOT_CONVERT

Data cannot be converted for this queue.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the code page being used.

CWB_API_ERROR

General API failure.

CWB_INVALID_HANDLE

Invalid system handle.

Usage: This function requires that you previously have issued **cwbCO_CreateSystem**.

cwbdQ_Peek

Purpose: Read a record from the iSeries data queue object that is identified by the specified handle. When a record is peeked from a data queue, it remains in the data queue. You may wait for a record if the data queue is empty by specifying a wait time from 0 to 99,999 or forever (-1). A wait time of zero will return immediately if there is no data in the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbdQ_Peek(  
    cwbdQ_QueueHandle queueHandle,  
    cwbdQ_Data data,  
    signed long waitTime,  
    cwbdSV_ErrHandle errorHandler);
```

Parameters:

cwbdQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbdQ_Open** API. This identifies the iSeries data queue object.

cwbdQ_Data data - input

The data object to be read from the iSeries data queue.

signed long waitTime - input

Length of time in seconds to wait for data, if the data queue is empty. A wait time of -1 indicates to wait forever.

cwbdSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the **cwbdSV_CreateErrHandle** API. The messages may be retrieved through the **cwbdSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_TIME

Invalid wait time.

CWBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

CWBDQ_INVALID_SEARCH

Invalid search order.

CWBDQ_DATA_TRUNCATED

Data truncated.

CWBDQ_TIMED_OUT

Wait time expired and no data returned.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_QUEUE_DESTROYED

Queue was destroyed.

CWBDQ_CANNOT_CONVERT

Unable to convert data.

Usage: This function requires that you have previously issued **cwbdQ_Open** or **cwbdQ_OpenEx** and **cwbdQ_CreateData**.

cwbDQ_Read

Purpose: Read a record from the iSeries data queue object that is identified by the specified handle. When a record is read from a data queue, it is removed from the data queue. You may wait for a record if the data queue is empty by specifying a wait time from 0 to 99,999 or forever (-1). A wait time of zero will return immediately if there is no data in the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Read(  
    cwbDQ_QueueHandle queueHandle,  
    cwbDQ_Data data,  
    long waitTime,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the `cwbDQ_Open` function. This identifies the iSeries data queue object.

cwbDQ_Data data - input

The data object to be read from the iSeries data queue.

long waitTime - input

Length of time in seconds to wait for data, if the data queue is empty. A wait time of -1 indicates to wait forever.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_TIME

Invalid wait time.

CWBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

CWBDQ_INVALID_SEARCH

Invalid search order.

CWBDQ_DATA_TRUNCATED

Data truncated.

CWBDQ_TIMED_OUT

Wait time expired and no data returned.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_QUEUE_DESTROYED

Queue was destroyed.

CWBDQ_CANNOT_CONVERT

Unable to convert data.

Usage: This function requires that you have previously issued `cwbDQ_Open` and `cwbDQ_CreateData`.

cwbDQ_SetAuthority

Purpose: Set the attribute for the authority that other users will have to the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetAuthority(  
                                cwbDQ_Attr      queueAttributes,  
                                unsigned short   authority);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

unsigned short authority - input

Authority that other users on the iSeries system have to access the data queue. Use one of the following defined types for authority:

- CWBDQ_ALL
- CWBDQ_EXCLUDE
- CWBDQ_CHANGE
- CWBDQ_USE
- CWBDQ_LIBCRTAUT

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

CWBDQ_INVALID_AUTHORITY

Invalid queue authority.

Usage: None

cwbDQ_SetConvert

Purpose: Set the convert flag. If the flag is set, all data being written will be converted from PC CCSID (for example, ASCII) to host CCSID (for example, EBCDIC), and all data being read will be converted from host CCSID (for example, EBCDIC) to PC CCSID (for example, ASCII). Default behavior is no conversion of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetConvert(  
    cwbDQ_Data data,  
    cwb_Boolean convert);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

cwb_Boolean convert - input

Flag indicating if data written to and read from the queue will be CCSID converted.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

Usage: None

cwbDQ_SetData

Purpose: Set the data and data length attributes of the data object. The default is to have no data with zero length. This function will make a copy of the data.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetData(  
    cwbDQ_Data data,  
    unsigned char *dataBuffer,  
    unsigned long dataLength);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned char * dataBuffer - input

Pointer to the data. The data may contain embedded NULLS, so it is not an ASCIIZ string.

unsigned long dataLength - input

Length of the data in bytes.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

CWBDQ_BAD_DATA_LENGTH

Length of data is not correct.

Usage: Use this function if you want to write a small amount of data or you do not want to manage the memory for the data in your application. Data will be copied and this may affect your application's performance.

cwbDQ_SetDataAddr

Purpose: Set the data and data length attributes of the data object. The default is to have no data with zero length. This function will not copy the data.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetDataAddr(  
    cwbDQ_Data      data,  
    unsigned char   *dataBuffer,  
    unsigned long   dataLength);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned char * dataBuffer - input

Pointer to the data. The data may contain embedded NULLS, so it is not an ASCII string.

unsigned long dataLength - input

Length of the data in bytes.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

CWBDQ_BAD_DATA_LENGTH

Length of data is not correct.

Usage: This function is better for large amounts of data, or if you want to manage memory in your application. Data will not be copied so performance will be improved.

cwbDQ_SetDesc

Purpose: Set the attribute for the description of the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetDesc(  
                                cwbDQ_Attr    queueAttributes,  
                                char           *description);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

char * description - input

Pointer to an ASCIIZ string that contains the description for the data queue. The maximum length for the description is 50 characters.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or null pointer.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

CWBDQ_INVALID_QUEUE_TITLE

Queue title is too long.

Usage: None

cwbDQ_SetForceToStorage

Purpose: Set the attribute for whether records will be forced to auxiliary storage when they are enqueued.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetForceToStorage(  
    cwbDQ_Attr      queueAttributes,  
    cwb_Boolean     forceToStorage);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

cwb_Boolean forceToStorage - input

Boolean indicator of whether each record is forced to auxiliary storage when it is enqueued.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_DQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbdQ_SetKey

Purpose: Set the key and key length attributes of the data attributes. This is the key that is used for writing data to a keyed data queue. In addition to the search order, this key is used to read data from a keyed data queue. The default is to have no key with zero length; this is the correct value for a non-keyed (LIFO or FIFO) data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbdQ_SetKey(  
    cwbdQ_Data      data,  
    unsigned char   *key,  
    unsigned short  keyLength);
```

Parameters:

cwbdQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbdQ_CreateData**.

unsigned char * key - input

Pointer to the key. The key may contain embedded NULLS, so it is not an ASCII string.

unsigned short keyLength - input

Length of the key in bytes.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

CWBDQ_BAD_KEY_LENGTH

Length of key is not correct.

Usage: None

cwbDQ_SetKeySize

Purpose: Set the attribute for the key size in bytes.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetKeySize(  
    cwbDQ_Attr      queueAttributes,  
    unsigned short  keySize);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

unsigned short keySize - input

Size in bytes of the key. This value should be zero if the order is LIFO or FIFO, and between 1 and 256 for KEYED.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_KEY_LENGTH

Invalid key length.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_SetMaxRecLen

Purpose: Set the maximum record length for the data queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetMaxRecLen(  
    cwbDQ_Attr          queueAttributes,  
    unsigned long       maxRecordLength);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

unsigned long maxLength - input

Maximum length for a data queue record. This value must be between 1 and 31744.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

CWBDQ_INVALID_QUEUE_LENGTH

Invalid queue record length.

Usage: None

cwbdq_SetOrder

Purpose: Set the attribute for the queue order. If the order is CWBDQ_SEQ_LIFO, the last record written is the first record read (Last In First Out). If the order is CWBDQ_SEQ_FIFO, the first record written is the first record read (First In First Out). If the order is CWBDQ_SEQ_KEYED, the order in which records are read from the data queue depends on the value of the search order attribute of the data object and the key value specified for the **cwbdq_SetKey** API. If multiple records contain the key that satisfies the search order, a FIFO scheme is used among those records.

Syntax:

```
unsigned int CWB_ENTRY cwbdq_SetOrder(  
                                cwbdq_Attr      queueAttributes,  
                                unsigned short  order);
```

Parameters:

cwbdq_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbdq_CreateAttr**.

unsigned short order - input

Order in which new entries will be enqueued. Use one of the following defined types for order:

CWBDQ_SEQ_LIFO

CWBDQ_SEQ_FIFO

CWBDQ_SEQ_KEYED

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

CWBDQ_INVALID_ORDER

Invalid queue order.

Usage: None

cwbDQ_SetSearchOrder

Purpose: Set the search order of the open attributes. The default is no search order. If the **cwbDQ_SetKey** API is called, the search order is changed to equal. Use this API to set it to something else. The search order is used when reading or peeking a keyed data queue to identify the relationship between the key of the record to retrieve and the key value specified on the **cwbDQ_SetKey** API. If the data queue order attribute is not CWBDQ_SEQ_KEYED, this property is ignored.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetSearchOrder(  
    cwbDQ_Data data,  
    unsigned short searchOrder);
```

Parameters:

cwbDQ_Data data - input

Handle of the data object that was returned by a previous call to **cwbDQ_CreateData**.

unsigned short searchOrder - input

Order to use when reading from a keyed queue. Possible values are:

- CWBDQ_NONE
- CWBDQ_EQUAL
- CWBDQ_NOT_EQUAL
- CWBDQ_GT_OR_EQUAL
- CWBDQ_GREATER
- CWBDQ_LT_OR_EQUAL
- CWBDQ_LESS

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_INVALID_DATA_HANDLE

Invalid data handle.

CWBDQ_INVALID_SEARCH

Invalid search order.

Usage: None

cwbDQ_SetSenderID

Purpose: Set the attribute for whether information about the sender is kept with each record on the queue.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_SetSenderID(  
    cwbDQ_Attr          queueAttributes,  
    cwb_Boolean         senderID);
```

Parameters:

cwbDQ_Attr queueAttributes - input

Handle of the data queue attributes returned by a previous call to **cwbDQ_CreateAttr**.

cwb_Boolean senderID - input

Boolean indicator of whether information about the sender is kept with record on the queue.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_DQ_INVALID_ATTRIBUTE_HANDLE

Invalid attributes handle.

Usage: None

cwbDQ_Write

Purpose: Write a record to the iSeries data queue object that is identified by the specified handle. Writing with commit ON means that your application will not have control returned to it until after the message has been enqueued.

Syntax:

```
unsigned int CWB_ENTRY cwbDQ_Write(  
    cwbDQ_QueueHandle queueHandle,  
    cwbDQ_Data data,  
    cwb_Boolean commit,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDQ_QueueHandle queueHandle - input

Handle that was returned by a previous call to the **cwbDQ_Open** or **cwbDQ_OpenEx** functions. This identifies the iSeries data queue object.

cwbDQ_Data data - input

The data object to be written to the iSeries data queue.

cwb_Boolean commit - input

Boolean flag indicating if the data should be committed on write.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBDQ_BAD_DATA_LENGTH

Length of data is not correct.

CWBDQ_INVALID_MESSAGE_LENGTH

Invalid message length.

CWBDQ_INVALID_QUEUE_HANDLE

Invalid queue handle.

CWBDQ_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBDQ_CANNOT_CONVERT

Unable to convert data.

Usage: This function requires that you previously have issued **cwbDQ_Open** or **cwbDQ_OpenEx**, and **cwbDQ_CreateData**.

Example: Using Data Queues APIs

```
// Sample Data Queues application

#ifdef UNICODE
    #define _UNICODE
#endif
#include <windows.h>

// Include the necessary DQ Classes
#include <stdlib.h>
#include <iostream.h>
#include "cwbdq.h"

/*****/

void main()
{
    cwbdQ_Attr queueAttributes;
    cwbdQ_QueueHandle queueHandle;
    cwbdQ_Data queueData;

    // Create an attribute object
    if ( (queueAttributes = cwbdQ_CreateAttr()) == 0 )
        return;

    // Set the maximum record length to 100
    if ( cwbdQ_SetMaxRecLen(queueAttributes,
                            100) != 0 )

        return;

    // Set the order to First-In-First-Out
    if (cwbdQ_SetOrder(queueAttributes, CWBDQ_SEQ_FIFO) != 0 )
        return;

    // Create the data queue DTAQ in library QGPL on system SYS1
    if ( cwbdQ_Create(_TEXT("DTAQ"),
                     _TEXT("QGPL"),
                     _TEXT("SYSNAMEXXX"),
                     queueAttributes,
                     NULL) != 0 )

        return;

    // Delete the attributes
    if ( cwbdQ_DeleteAttr( queueAttributes ) != 0 )
        return;

    // Open the data queue
    if ( cwbdQ_Open(_TEXT("DTAQ"),
                   _TEXT("QGPL"),
                   _TEXT("SYSNAMEXXX"),
                   &queueHandle,
                   NULL) != 0 )

        return;

    // Create a data object
    if ( (queueData = cwbdQ_CreateData()) == 0 )
        return;
}
```

```

// Set the data length and the data
if ( cwbDQ_SetData(queueData, (unsigned char*)"Test Data!", 10) != 0 )
    return;

// Write the data to the data queue
if ( cwbDQ_Write(queueHandle, queueData, CWB_TRUE, NULL) != 0 )
    return;

// Delete the data object
if ( cwbDQ_DeleteData(queueData) != 0 )
    return;

// Close the data queue
if ( cwbDQ_Close(queueHandle) != 0 )
    return;

}

```

iSeries Access for Windows Data Transformation and National LanguageSupport (NLS) APIs

“iSeries Access for Windows data transformation APIs”

iSeries Access for Windows **data transformation** application programming interfaces (APIs) enable your client/server applications to transform numeric data between iSeries server and PC formats. Transformation may be required when you send and receive numeric data to and from the iSeries server. Data transformation APIs support transformation of many numeric formats.

“iSeries Access for Windows national language support (NLS) APIs” on page 228

iSeries Access for Windows **national language support** APIs enable your applications to get and save (query and change) the iSeries Access for Windows settings that are relevant to national language support. You can add convenient functions into your iSeries Access for Windows applications, including the capability to:

- Select from a list of installed national languages.
- Convert character data from one code page to another. This permits computers that use different code pages, such as personal computers and the iSeries server, to share information.
- Automatically replace the translatable text (caption and control names) within dialog boxes. This expands the size of the controls according to the text that is associated with them. The size of the dialog-box frame also is adjusted automatically.

iSeries Access for Windows data transformation APIs

iSeries Access for Windows data transformation APIs required files:

Header file	Import library	Dynamic Link Library
cwbdt.h	cwbapi.lib	cwbdt.dll

Programmer’s Toolkit:

The Programmer’s Toolkit provides data transformation documentation, access to the cwbdt.h header file, and links to sample programs. To access this information, open the Programmer’s Toolkit and select **Data Manipulation** → **C/C++ APIs**.

iSeries Access for Windows data transformation APIs topics:

- **iSeries Access for Windows data transformation APIs listing**
- “Example: Using data transformation APIs” on page 228

Related topics:

- “iSeries system name formats for ODBC Connection APIs” on page 12
- “OEM, ANSI, and Unicode considerations” on page 12

iSeries Access for Windows data transformation API listing

Note: iSeries Access for Windows data transformation APIs that accept strings are provided in Unicode versions. In these APIs, “ASCII” is replaced by “Wide” (for example, `cwbDT_ASCII11ToBin4` has a Unicode version: `cwbDT_Wide11ToBin4`). These APIs are indicated in the table that follows. The Unicode versions have different syntax, parameters and return values than their ASCII counterparts.

See “OEM, ANSI, and Unicode considerations” on page 12 and the `cwbdt.h` header file for details.

iSeries Access for Windows data transformation API	Unicode version
<code>cwbDT_ASCII11ToBin4</code>	<code>cwbDT_Wide11ToBin4</code>
<code>cwbDT_ASCII6ToBin2</code>	<code>cwbDT_Wide6ToBin2</code>
<code>cwbDT_ASCIIpackedToPacked</code>	None
<code>cwbDT_ASCIItoHex</code>	<code>cwbDT_WideToHex</code>
<code>cwbDT_ASCIItoPacked</code>	<code>cwbDT_WideToPacked</code>
<code>cwbDT_ASCIItoZoned</code>	<code>cwbDT_WideToZoned</code>
<code>cwbDT_ASCIIzonedToZoned</code>	None
<code>cwbDT_Bin2toASCII6</code>	<code>cwbDT_Bin2toWide6</code>
<code>cwbDT_Bin2toBin2</code>	None
<code>cwbDT_Bin4toASCII11</code>	<code>cwbDT_Bin4toWide11</code>
<code>cwbDT_Bin4toBin4</code>	None
<code>cwbDT_EBCDICtoEBCDIC</code>	None
<code>cwbDT_HextoASCII</code>	<code>cwbDT_HextoWide</code>
<code>cwbDT_PackedtoASCII</code>	<code>cwbDT_PackedtoWide</code>
<code>cwbDT_PackedtoASCIIPacked</code>	None
<code>cwbDT_PackedtoPacked</code>	None
<code>cwbDT_ZonedtoASCII</code>	<code>cwbDT_ZonedtoWide</code>
<code>cwbDT_ZonedtoASCIIzoned</code>	None
<code>cwbDT_ZonedtoZoned</code>	None

cwbDT_ASCII11ToBin4:

Purpose: Translates (exactly) 11 ASCII numeric characters to a 4-byte integer stored most significant byte first. (The source string is not expected to be zero-terminated.) This function can be used for translating ASCII numeric data to the iSeries integer format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ASCII11ToBin4(  
    char *target,  
    char *source);
```

Parameters:

char * target - output

Pointer to the target (4 byte integer).

char * source - input

Pointer to the source (11 byte ASCII).

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

CWB_BUFFER_OVERFLOW

Overflow error.

other Offset of the first untranslated character plus one.

Usage: The target data will be stored with the Most Significant Byte first. This is the format that the iSeries server uses and is the opposite of the format that is used by the Intel x86 processors. Valid formats for the ASCII source data are as follows:

[blankspaces][sign][blankspaces][digits] or
[sign][blankspaces][digits][blankspaces]

Examples:

```
" + 123"  
"- 123 "  
" +123 "  
" 123"  
" -123"  
"+123 "
```

cwbDT_ASCII6ToBin2:

Purpose: Translates (exactly) 6 ASCII numeric characters to a 2-byte integer stored most significant byte first. (The source string is not expected to be zero-terminated.) This function can be used for translating ASCII numeric data to the iSeries integer format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ASCII6ToBin2(  
    char *target,  
    char *source);
```

Parameters:

char * target - output

Pointer to the target (2 byte integer).

char * source - input

Pointer to the source (6 byte ASCII).

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

CWB_BUFFER_OVERFLOW

Overflow error.

other Offset of the first untranslated character plus one.

Usage: The target data will be stored with the Most Significant Byte first. This is the format that the iSeries server uses and is the opposite of the format that is used by Intel x86 processors. Valid formats for the ASCII source data are as follows:

[blankspaces][sign][blankspaces][digits] or
[sign][blankspaces][digits][blankspaces]

Examples:

```
" + 123"  
"- 123 "  
"+123 "  
" 123"  
"-123"  
"+123 "
```


cwbDT_ASCIIpackedToPacked:

Purpose: Translates data from ASCII packed format to packed decimal. This function can be used for translating data from ASCII files to the iSeries system format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ASCIIpackedToPacked(  
    char          *target,  
    char          *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. This function checks that each half-byte of the packed decimal data is in the range of 0 to 9. The only exception is the last half-byte which contains the sign indicator (which can be 0x3 or 0xb).

cwbDT_ASCIItoHex:

Purpose: Translates data from ASCII (hex representation) to binary. One byte is stored in the target for each two bytes in the source.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ASCIItoHex(  
    char      *target,  
    char      *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source (ASCII hex) data.

unsigned long length - input

Number of bytes of source data to translate/2.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: For 'length' bytes of source data 'length'/2 bytes of target data will be stored. The caller must make sure that there is adequate space to hold the target information.

cwbDT_ASCIItoPacked:

Purpose: Translates ASCII numeric data to packed decimal format. This function can be used for translating ASCII text data for use on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ASCIItoPacked(  
    char          *target,  
    char          *source,  
    unsigned long length,  
    unsigned long decimalPosition);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data. Must be zero terminated.

unsigned long length - input

Number of bytes of target data to translate.

unsigned long decimalPosition - input

Position of the decimal point.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

CWB_BUFFER_OVERFLOW

Overflow error.

CWB_NOT_ENOUGH_MEMORY

Unable to allocate temporary memory.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. The sign half-byte will be set to 0xd to indicate a negative number and hex 0xc to indicate a positive number. $0 \leq \text{decimalPosition} < (\text{length} * 2)$. Valid formats for the ASCII numeric data are as follows:

```
[blankspaces][sign][blankspaces][digits] or  
[sign][blankspaces][digits][blankspaces] or  
[sign][digits][.digits][blankspaces] or  
[blankspaces][sign][digits][.digits][blankspaces]
```

Examples:

```
" + 123\0"  
"- 123 \0"  
" +123 \0"  
" 123\0"  
" -12.3\0"  
"+1.23 \0"
```

cwbDT_ASCIItoZoned:

Purpose: Translates ASCII numeric data to EBCDIC zoned decimal format. This function can be used for translating ASCII text data for use on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ASCIItoZoned(  
    char *target,  
    char *source,  
    unsigned long length,  
    unsigned long decimalPosition);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data. Must be zero terminated.

unsigned long length - input

Number of bytes of target data to translate.

unsigned long decimalPosition - input

Position of the decimal point.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

CWB_BUFFER_OVERFLOW

Overflow error.

CWB_NOT_ENOUGH_MEMORY

Unable to allocate temporary memory.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the information. The sign half-byte will be set to 0xd to indicate a negative number and hex 0xc to indicate a positive number. $0 \leq \text{decimalPosition} \leq \text{length}$. Valid formats for the ASCII numeric data are as follows:

```
[blankspaces][sign][blankspaces][digits] or  
[sign][blankspaces][digits][blankspaces] or  
[sign][digits][.digits][blankspaces] or  
[blankspaces][sign][digits][.digits][blankspaces]
```

Examples:

```
" + 123\0"  
"- 123 \0"  
" +123 \0"  
" 123\0"  
" -12.3\0"  
"+1.23 \0"
```

cwbDT_ASCIIzonedToZoned:

Purpose: Translates data from ASCII zoned decimal format to EBCDIC zoned decimal. This function can be used for translating data from ASCII files for use on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ASCIIzonedToZoned(  
    char          *target,  
    char          *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The left half of each byte (0x3) in the ASCII zoned decimal format will be converted to 0xf in the left half-byte of the EBCDIC zoned data except for the last byte (sign). This function checks that the left half of each byte in the ASCII zoned decimal data must be 0x3 except for the last byte. The high half of the last byte must be 0x3 or 0xb. The right half of each byte in the ASCII zoned decimal data must be in the range 0-9.

cwbDT_Bin2ToASCII6:

Purpose: Translates a 2-byte integer stored most significant byte first to (exactly) 6 ASCII numeric characters. (The target will not be zero terminated.) This function can be used for translating numeric data from an iSeries server to ASCII.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_Bin2ToASCII6(  
    char *target,  
    char *source);
```

Parameters:

char * target - output

Pointer to the target (6 byte) area.

char * source - input

Pointer to the source (2 byte integer).

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

Usage: The source data is assumed to be stored with the Most significant Byte first. This is the format that the iSeries server uses and is the opposite of the format used by the Intel x86 processes.

cwbDT_Bin2ToBin2:

Purpose: Reverses the order of bytes in a 2-byte integer. This function can be used for translating a 2-byte integer to or from the iSeries server format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_Bin2ToBin2(  
    char *target,  
    char *source);
```

Parameters:

char * target - output

Pointer to the target (2 byte integer).

char * source - input

Pointer to the source (2 byte integer).

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

Usage: The source data and the target data must not overlap. The following example shows the result of the translation:

Source data: 0x1234

Target data: 0x3412

cwbDT_Bin4ToASCII11:

Purpose: Translates a 4-byte integer stored most significant byte first to (exactly) 11 ASCII numeric characters. (The target will not be zero terminated.) This function can be used for translating numeric data from an iSeries server to ASCII.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_Bin4ToASCII11(  
    char *target,  
    char *source );
```

Parameters:

char * target - output

Pointer to the target (11 byte) area.

char * source - input

Pointer to the source (4 byte integer).

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

Usage: The source data is assumed to be stored with the Most Significant Byte first. This is the format that the iSeries server uses and is the opposite of the format used by the Intel x86 processors.

cwbDT_Bin4ToBin4:

Purpose: Reverses the order of bytes in a 4-byte integer. This function can be used for translating a 4-byte integer to or from the iSeries server format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_Bin4ToBin4(  
    char *target,  
    char *source);
```

Parameters:

char * target - output

Pointer to the target (4 byte integer).

char * source - input

Pointer to the source (4 byte integer).

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

Usage: The source data and the target data must not overlap. The following example shows the result of the translation:

Source data: 0x12345678

Target data: 0x78563412

cwbDT_EBCDICToEBCDIC:

Purpose: 'Translates' (copies unless character value less than 0x40 is encountered) EBCDIC data to EBCDIC.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_EBCDICToEBCDIC(  
    char          *target,  
    char          *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of target data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information.

cwbDT_HexToASCII:

Purpose: Translates binary data to the ASCII hex representation. Two ASCII characters are stored in the target for each byte of source data.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_HexToASCII(  
    char      *target,  
    char      *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target (ASCII hex) data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

Usage: For 'length' bytes of source data 'length'*2 bytes of target data will be stored. The caller must make sure that there is adequate space to hold the target information.

cwbDT_PackedToASCII:

Purpose: Translates data from packed decimal format to ASCII numeric data. This function can be used for translating data from the the iSeries server for use in ASCII text format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_PackedToASCII(  
    char      *target,  
    char      *source,  
    unsigned long length,  
    unsigned long decimalPosition);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

unsigned long decimalPosition - input

Position of the decimal point.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. This function checks that each half-byte of the packed decimal data is in the range of 0 to 9. The only exception is the last half-byte which contains the sign indicator. $0 \leq \text{decimalPosition} < (\text{length} * 2)$.

cwbDT_PackedToASCIIPacked:

Purpose: Translates data from packed decimal format to ASCII packed format. This function can be used for translating data from the iSeries server for use in ASCII format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_PackedToASCIIPacked(  
    char          *target,  
    char          *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. This function checks that each half-byte of the packed decimal data is in the range of 0 to 9. The only exception is the last half-byte which contains the sign indicator (which can be 0-9, 0xd, or 0xb).

cwbDT_PackedToPacked:

Purpose: Translates packed decimal data to packed decimal. This function can be used for transferring data from the iSeries system to no-conversion files and back.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_PackedToPacked(  
    char          *target,  
    char          *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. This function checks that each half-byte of the packed decimal data is in the range of 0 to 9. The only exception is the last half-byte which contains the sign indicator.

cwbDT_ZonedToASCII:

Purpose: Translates EBCDIC zoned decimal data to ASCII numeric format. This function can be used for translating data from the iSeries server for use in ASCII text format.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ZonedToASCII(  
    char          *target,  
    char          *source,  
    unsigned long length,  
    unsigned long decimalPosition);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

unsigned long decimalPosition - input

Position of the decimal point.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

CWB_BUFFER_OVERFLOW

Overflow error.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. The high half of the last byte of the zoned data indicates the sign of the number. If the high half-byte is 0xb or 0xd, then a negative number is indicated. Any other value indicates a positive number. This function checks that the high half of each byte of zoned data must be 0xf except for the last byte. The low half of each byte of zoned data must be in the range 0-9. $0 \leq \text{decimalPosition} < \text{length}$.

cwbDT_ZonedToASCIIZoned:

Purpose: Translates data from EBCDIC zoned decimal format to ASCII zoned decimal format. This function can be used for translating data from the iSeries server for use in ASCII files.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ZonedToASCIIZoned(  
    char      *target,  
    char      *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. The left half-byte (0xf) in the EBCDIC zoned decimal data will be converted to 0x3 in the left half-byte of the ASCII zoned decimal data except for the last byte (sign). The high half of the last byte of the EBCDIC zoned decimal data indicates the sign of the number. If the high half-byte is 0xb or 0xb then a negative number is indicated, any other value indicates a positive number. This function checks that the high half of each byte of EBCDIC zoned decimal data must be 0xf except for the last byte. The low half of each byte of EBCDIC zoned decimal data must be in the range 0-9.

cwbDT_ZonedToZoned:

Purpose: Translates data from zoned decimal format to zoned decimal. This function can be used for translating data from the iSeries server for use in no-conversion files and vice-versa.

Syntax:

```
unsigned int CWB_ENTRY cwbDT_ZonedToZoned(  
    char          *target,  
    char          *source,  
    unsigned long length);
```

Parameters:

char * target - output

Pointer to the target data.

char * source - input

Pointer to the source data.

unsigned long length - input

Number of bytes of source data to translate.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_INVALID_POINTER

NULL pointer was passed by caller.

other Offset of the first untranslated character plus one.

Usage: The caller must make sure that there is adequate space to hold the target information. The high half of the last byte of the zoned data indicates the sign of the number. If the high half-byte is 0xb or 0xb then a number is indicated, any other value indicates a positive number. This function checks that the high half of each byte of zoned data must be 0xf except for the last byte. The low half of each byte of zoned data must be in the range 0-9.

Example: Using data transformation APIs

```
/******  
/* Sample Data Transform Program using cwBDT_Bin4ToBin4 to reverse */  
/* the order of bytes in a 4-byte integer. */  
/******  
  
#include <iostream.h>  
#include "cwBDT.h"  
  
void main()  
{  
    unsigned int returnCode;  
    long source,  
        target;  
  
    cout << "Enter source number:\n";  
  
    while (cin >> source) {  
        cout << "Source in Dec = " << dec << source;  
        cout << "\nSource in Hex = " << hex << source << '\n';  
        if (((returnCode = cwBDT_Bin4ToBin4((char *)&target,(char *)&source)) == CWB_OK)) {  
            cout << "Target in Dec = " << dec << target;  
            cout << "\nTarget in Hex = " << hex << target << '\n';  
        } else {  
            cout << "Conversion failed, Return code = " << returnCode << '\n' ;  
        }; /* endif */  
        cout << "\nEnter source number:\n";  
  
    }; /* endwhile */  
}
```

iSeries Access for Windows national language support (NLS) APIs

iSeries servers support many national languages, through national language support (NLS). NLS allows users to work on an iSeries system in the language of their choice. The iSeries system also ensures that the data that is sent to and received from the system appears in the form and order that is expected. By supporting many different languages, the system operates as intended, from both a linguistic and a cultural point of view.

All iSeries systems use a common set of program code, regardless of which language you use on the system. For example, the program code on a U.S. English iSeries system and the program code on a Spanish iSeries system are identical. Different sets of textual data are used, however, for different languages. Textual data is a collective term for menus, displays, lists, prompts, options, on-line help information, and messages. This means that you see *Help* for the description of the function key for on-line help information on a U.S. English system, while you see *Ayuda* on a Spanish system. Using the same program code with different sets of textual data allows the iSeries system to support more than one language on a single system.

Note: It is essential to build national language support considerations into the design of the program right from the start. It is much harder to add NLS or DBCS support after a program has been designed or coded.

iSeries Access for Windows NLS APIs required files:

NLS API type	Header file	Import library	Dynamic Link Library
General	cwbnl.h	cwbapi.lib	cwbnl.dll
Conversion	cwbnlcnv.h		cwbnl1.dll
Dialog-box	cwbnldlg.h		cwbnldlg.dll

Programmer's Toolkit:

The Programmer's Toolkit provides NLS documentation, access to the NLS APIs header files, and links to sample programs. To access this information, open the Programmer's Toolkit and select **Data Manipulation** → **C/C++ APIs**.

iSeries Access for Windows NLS APIs topics:

- "Coded character sets"
- **iSeries Access for Windows NLS APIs listing**
- "Example: iSeries Access for Windows NLS APIs" on page 259

Related topics:

- "iSeries system name formats for ODBC Connection APIs" on page 12
- "OEM, ANSI, and Unicode considerations" on page 12

Coded character sets

Graphic characters are printable or displayable symbols, such as letters, numbers, and punctuation marks. A collection of graphic characters is called a *graphic-character set*, and often simply a *character set*. Each language requires its own graphic-character set to be printed or displayed properly. Characters are encoded according to a *code page*, which is a table that assigns graphic and control characters to specific values called *code points*.

Code pages are classified into many types according to the encoding scheme. Two important encoding schemes for iSeries Access are the Host and PC code pages. Unicode also is becoming an important encoding scheme. Unicode is a 16-bit worldwide character encoding scheme that is gaining popularity on both the Host and the personal computer.

- Host code pages are encoded in accordance with IBM Standard of Extended BCD Interchange Code (EBCDIC) and usually used by S/390 and iSeries servers.
- PC Code pages are encoded based on ANSI X3.4, ASCII and usually used by IBM Personal Computers.

iSeries Access for Windows NLS APIs listing

The iSeries Access for Windows national language support application programming interfaces (APIs) are listed in alphabetical order. They provide necessary information for their use. They are grouped into three functional categories:

- iSeries Access for Windows general national language support APIs
- iSeries Access for Windows conversion national language support APIs
- iSeries Access for Windows dialog-box national language support APIs

iSeries Access for Windows general NLS APIs list: iSeries Access for Windows is translated into many languages. One or more of these languages can be installed on the personal computer. The following iSeries Access for Windows general NLS APIs allow an application to:

- Get a list of installed languages
- Get the current language setting
- Save the language setting

`cwbNL_FindFirstLang`

`cwbNL_FindNextLang`

`cwbNL_GetLang`

`cwbNL_GetLangName`

cwbNL_GetLangPath
cwbNL_SaveLang

cwbnl_FindFirstLang:

Purpose: Returns the first available language.

Syntax:

```
unsigned int CWB_ENTRY cwbnl_FindFirstLang(  
    char          *mriBasePath,  
    char          *resultPtr,  
    unsigned short resultLen,  
    unsigned short *requiredLen,  
    unsigned long  *searchHandle,  
    cwbsv_ErrHandle errorHandler);
```

Parameters:

char * mriBasePath - input

Pointer to the mriBasePath, e.g. C:\Program Files\IBM\ClientAccess/400 If NULL, the mriBasePath of the ClientAccess/400 product is used.

char * resultPtr - output

Pointer to the buffer to contain the result.

unsigned short resultLen - input

Length of the result buffer. Recommended size is CWBNL_MAX_LANG_SIZE.

unsigned short * requiredLen - output

Actual length of the result. If requiredLen > resultLen, the return value will be CWB_BUFFER_OVERFLOW.

unsigned long * searchHandle - output

Search handle to be passed on subsequent calls to **cwbnl_FindNextLang**.

cwbsv_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbsv_CreateErrHandle()** API. The messages may be retrieved through the **cwbsv_GetErrText()** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_FILE_NOT_FOUND

File not found.

CWB_PATH_NOT_FOUND

Path not found.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

Usage: The result buffer will contain a language.

cwbNL_FindNextLang:

Purpose: Returns the next available language.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_FindNextLang(  
    char          *resultPtr,  
    unsigned short resultLen,  
    unsigned short *requiredLen,  
    unsigned long  *searchHandle,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

char * resultPtr - output

Pointer to the buffer to contain the result.

unsigned short resultLen - input

Length of the result buffer. Recommended size is CWBNL_MAX_LANG_SIZE.

unsigned short * requiredLen - output

Actual length of the result. If requiredLen > resultLen, the return value will be CWB_BUFFER_OVERFLOW.

unsigned long * searchHandle - output

Search handle to be passed on subsequent calls to **cwbNL_FindNextLang**.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle()** API. The messages may be retrieved through the **cwbSV_GetErrText()** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NO_MORE_FILES

No more files are found.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

Usage: The result buffer will contain a language.

cwbNL_GetLang:

Purpose: Get the current language setting.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_GetLang(  
    char          *mriBasePath,  
    char          *resultPtr,  
    unsigned short resultLen,  
    unsigned short *requiredLen,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

char * mriBasePath - input

Pointer to the mriBasePath, e.g. C:\Program Files\IBM\ClientAccess/400. If NULL, the mriBasePath of the ClientAccess/400 product is used.

char * resultPtr - output

Pointer to the buffer to contain the result.

unsigned short resultLen - input

Length of the result buffer. Recommended size is CWBNL_MAX_LANG_SIZE.

unsigned short * requiredLen - output

Actual length of the result. If requiredLen > resultLen, the return value will be CWB_BUFFER_OVERFLOW.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle()** API. The messages may be retrieved through the **cwbSV_GetErrText()** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_BUFFER_OVERFLOW

Buffer too small to contain result.

Usage: The result buffer will contain the name of the language subdirectory. This language subdirectory contains the language-specific files. This language subdirectory name also can be passed to **cwbNL_GetLangName**.

cwbNL_GetLangName:

Purpose: Return the descriptive name of a language setting.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_GetLangName(  
    char          *lang,  
    char          *resultPtr,  
    unsigned short resultLen,  
    unsigned short *requiredLen,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

char * lang - input

Address of the ASCIIZ string representing the language.

char * resultPtr - output

Pointer to the buffer to contain the result.

unsigned short resultLen - input

Length of the result buffer. Recommended size is CWBNL_MAX_NAME_SIZE.

unsigned short * requiredLen - output

Actual length of the result. If requiredLen > resultLen, the return value will be CWB_BUFFER_OVERFLOW.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle()** API. The messages may be retrieved through the **cwbSV_GetErrText()** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

Usage: The language must be a value returned from one of the following APIs:

```
cwbNL_GetLang  
cwbNL_FindFirstLang  
cwbNL_FindNextLang
```


cwbNL_GetLangPath:

Purpose: Return the complete path for language files.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_GetLangPath(  
    char          *mriBasePath,  
    char          *resultPtr,  
    unsigned short resultLen,  
    unsigned short *requiredLen,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

char * mriBasePath - input

Pointer to the mriBasePath, for example C:\Program Files\IBM\ClientAccess/400. If NULL, the mriBasePath of the ClientAccess/400 product is used.

char * resultPtr - output

Pointer to the buffer to contain the result.

unsigned short resultLen - input

Length of the result buffer. Recommended size is CWBNL_MAX_PATH_SIZE.

unsigned short * requiredLen - output

Actual length of the result. If requiredLen > resultLen, the return value will be CWB_BUFFER_OVERFLOW.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle()** API. The messages may be retrieved through the **cwbSV_GetErrText()** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_PATH_NOT_FOUND

Path not found.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

Usage: The result buffer will contain the complete path of the language subdirectory. Language files should be loaded from this path.

cwbNL_SaveLang:

Purpose: Save the language setting in the product registry.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_SaveLang(  
    char *lang,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

char * lang - input

Address of the ASCII string representing the language.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle()** API. The messages may be retrieved through the **cwbSV_GetErrText()** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: The language must be a value returned from one of the following APIs:

cwbNL_GetLang
cwbNL_FindFirstLang
cwbNL_FindNextLang

The following APIs are affected by this call:

cwbNL_GetLang
cwbNL_GetLangPath

iSeries Access for Windows conversion NLS APIs list: The following iSeries Access for Windows conversion NLS APIs allow applications to:

- Convert character data from one code page to another
- Determine the current code page setting
- Determine the last CCSID setting
- Convert code page values to and from code character set identifiers (CCSID)

`cwbNL_CCSIDToCodePage`

`cwbNL_CodePageToCCSID`

`cwbNL_Convert`

`cwbNL_ConvertCodePages`

`cwbNL_CreateConverter`

`cwbNL_DeleteConverter`

`cwbNL_GetCodePage`

`cwbNL_GetANSIcodePage`

`cwbNL_GetHostCCSID`

cwbNL_CCSIDToCodePage:

Purpose: Map CCSIDs to code pages.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_CCSIDToCodePage(  
    unsigned long  CCSID,  
    unsigned long *codePage,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

unsigned long CCSID - input

CCSID to convert to a code page.

unsigned long * codePage - output

The resulting code page.

cwbSV_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved with the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: None

cwbNL_CodePageToCCSID:

Purpose: Map code pages to CCSIDs.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_CodePageToCCSID(  
    unsigned long codePage,  
    unsigned long *CCSID,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

unsigned long codePage - input

Code page to convert to a CCSID.

unsigned long * CCSID - output

The resulting CCSID.

cwbSV_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved with the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: None

cwbnl_Convert:

Purpose: Convert strings by using a previously opened converter.

Syntax:

```
unsigned int CWB_ENTRY cwbnl_Convert(  
    cwbnl_Converter theConverter,  
    unsigned long   sourceLength,  
    unsigned long   targetLength,  
    char            *sourceBuffer,  
    char            *targetBuffer,  
    unsigned long   *numberOfErrors,  
    unsigned long   *firstErrorIndex,  
    unsigned long   *requiredLen,  
    cwbsv_ErrHandle errorHandle);
```

Parameters:

cwbnl_Converter theConverter - output

Handle to the previously opened converter.

unsigned long sourceLength - input

Length of the source buffer.

unsigned long targetLength - input

Length of the target buffer. If converting from an ASCII code page that contains DBCS characters, note that the resulting data could contain shift-out and shift-in bytes. Therefore, the targetBuffer may need to be larger than the sourceBuffer.

char *sourceBuffer - input

Buffer containing the data to convert.

char *targetBuffer - output

Buffer to contain the converted data.

unsigned long *numberOfErrors - output

Contains the number of characters that could not be converted properly.

unsigned long *firstErrorIndex - output

Contains the offset of the first character in the source buffer that could not be converted properly.

unsigned long *requiredLen - output

Actual length of the result. If requiredLen > resultLen, the return value will be CWB_BUFFER_OVERFLOW.

cwbsv_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **cwbsv_CreateErrHandle** API. The messages may be retrieved with the **cwbsv_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

Usage: None

cwbNL_ConvertCodePages:

Purpose: Convert strings from one code page to another. This API combines the following three converter APIs for the default conversion:

- `cwbNL_CreateConverter`
- `cwbNL_Convert`
- `cwbNL_DeleteConverter`

Syntax:

```
unsigned int CWB_ENTRY cwbNL_ConvertCodePages(  
    unsigned long    sourceCodePage,  
    unsigned long    targetCodePage,  
    unsigned long    sourceLength,  
    unsigned long    targetLength,  
    char             *sourceBuffer,  
    char             *targetBuffer,  
    unsigned long    *numberOfErrors,  
    unsigned long    *positionOfFirstError,  
    unsigned long    *requiredLen,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

unsigned long sourceCodePage - input

Code page of the data in the source buffer.

unsigned long targetCodePage - input

Code page to which the data should be converted.

unsigned long sourceLength - input.

Length of the source buffer

unsigned long targetLength - input.

Length of the target buffer

char *sourceBuffer - input

Buffer containing the data to convert.

char *targetBuffer - output

Buffer to contain the converted data.

unsigned long *numberOfErrors - output

Contains the number of characters that could not be converted properly.

unsigned long *positionOfFirstError - output

Contains the offset of the first character in the source buffer that could not be converted properly.

unsigned long *requiredLen - output

Actual length of the result. If `requiredLen > resultLen`, the return value will be `CWB_BUFFER_OVERFLOW`.

cwbSV_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **`cwbSV_CreateErrHandle`** API. The messages may be retrieved with the **`cwbSV_GetErrText`** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWBNL_ERR_CNV_UNSUPPORTED

An error occurred while attempting to convert the characters. No conversion was done. The most common reason is that a conversion table is missing. Conversion tables are either installed with iSeries Access for Windows, or retrieved from the default iSeries system when needed. There may have been some problem communicating with the default iSeries system.

CWBNL_ERR_CNV_ERR_STATUS

This return code is used to indicate that while the requested conversion is supported, and the conversion completed, there were some characters that did not convert properly. Either the source buffer contained null characters, or the characters do not exist in the target code page. Applications can choose to ignore this return code or treat it as a warning.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: The following values may be specified on the sourceCodePage and the targetCodePage parameters:

Value	Meaning
CWBNL_CP_UNICODE_F200	UCS2 Version 1.1 UNICODE
CWBNL_CP_UNICODE	UCS2 Current Version UNICODE
CWBNL_CP_AS400	AS/400 host code page
CWBNL_CP_CLIENT_OEM	OEM client code page
CWBNL_CP_CLIENT_ANSI	ANSI client code page
CWBNL_CP_CLIENT_UNICODE	UNICODE client code page
CWBNL_CP_UTF8	UCS transformation form, 8-bit format
CWBNL_CP_CLIENT	Generic client code page. Default is CWBNL_CP_CLIENT_OEM. CWBNL_CP_CLIENT is set to CWBNL_CP_CLIENT_ANSI when CWB_ANSI is defined, to CWBNL_CP_CLIENT_UNICODE when CWB_UNICODE is defined and to CWBNL_CP_CLIENT_OEM when CWB_OEM is defined.

cwbNL_CreateConverter:

Purpose: Create a **cwbNL_Converter** to be used on subsequent calls to **cwbNL_Convert()**.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_CreateConverter(  
    unsigned long    sourceCodePage,  
    unsigned long    targetCodePage,  
    cwbNL_Converter *theConverter,  
    cwbSV_ErrHandle  errorHandle,  
    unsigned long    shiftInShiftOutStatus,  
    unsigned long    padLength,  
    char             *pad);
```

Parameters:

unsigned long sourceCodePage - input

Code page of the source data.

unsigned long targetCodePage - input

Code page to which the data should be converted.

cwbNL_Converter * theConverter - output

The newly created converter.

cwbSV_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved with the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

unsigned long shiftInShiftOutStatus - input

Indicates whether the shift-in and shift-out bytes are part of the input or output data. 0 - False, no shift-in and shift-out bytes are part of the data string. 1 - True, shift-in and shift-out characters are part of the data string.

unsigned long padLength - input

Length of pad characters. 0 - No pad characters for this conversion request 1 - 1 byte of pad character. This is valid only if the target code page is either SBCS or DBCS code page 2 - 2 bytes of pad characters. This is valid only if the code page is not a single-byte code page.

char * pad - input

The character or characters for padding.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWBNL_ERR_CNV_UNSUPPORTED

An error occurred while attempting to convert the characters. No conversion was done. The most common reason is that a conversion table is missing. Conversion tables are either installed with iSeries Access for Windows, or retrieved from the default iSeries system when needed. There may have been some problem communicating with the default iSeries system.

CWBNL_ERR_CNV_ERR_STATUS

This return code is used to indicate that while the requested conversion is supported, and the

conversion completed, there were some characters that did not convert properly. Either the source buffer contained null characters, or the characters do not exist in the target code page. Applications can choose to ignore this return code or treat it as a warning.

CWBNL_ERR_CNV_INVALID_SISO_STATUS

Invalid SISO parameter.

CWBNL_ERR_CNV_INVALID_PAD_LENGTH

Invalid Pad Length parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: The following values may be specified on the `sourceCodePage` and the `targetCodePage` parameters:

Value	Meaning
<code>CWBNL_CP_UNICODE_F200</code>	UCS2 Version 1.1 UNICODE
<code>CWBNL_CP_UNICODE</code>	UCS2 Current Version UNICODE
<code>CWBNL_CP_AS400</code>	AS/400 host code page
<code>CWBNL_CP_CLIENT_OEM</code>	OEM client code page
<code>CWBNL_CP_CLIENT_ANSI</code>	ANSI client code page
<code>CWBNL_CP_CLIENT_UNICODE</code>	UNICODE client code page
<code>CWBNL_CP_UTF8</code>	UCS transformation form, 8-bit format
<code>CWBNL_CP_CLIENT</code>	Generic client code page. Default is <code>CWBNL_CP_CLIENT_OEM</code> . <code>CWBNL_CP_CLIENT</code> is set to <code>CWBNL_CP_CLIENT_ANSI</code> when <code>CWB_ANSI</code> is defined, to <code>CWBNL_CP_CLIENT_UNICODE</code> when <code>CWB_UNICODE</code> is defined and to <code>CWBNL_CP_CLIENT_OEM</code> when <code>CWB_OEM</code> is defined.

Instead of calling **`cwbnl_ConvertCodePages`** multiple times with the same code pages:

```
cwbnl_ConvertCodePages(850, 500, ...);  
cwbnl_ConvertCodePages(850, 500, ...);  
cwbnl_ConvertCodePages(850, 500, ...);
```

It is more efficient to create a converter and use it multiple times:

```
cwbnl_CreateConverter(850, 500, &conv, ...);  
cwbnl_Convert(conv, ...);  
cwbnl_Convert(conv, ...);  
cwbnl_Convert(conv, ...);  
cwbnl_DeleteConverter(conv, ...);
```

cwbNL_DeleteConverter:

Purpose: Delete a **cwbNL_Converter**.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_DeleteConverter(  
    cwbNL_Converter theConverter,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbNL_Converter theConverter - input

A previously created converter.

cwbSV_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle0** API. The messages may be retrieved with the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Invalid handle.

Usage: None

cwbNL_GetCodePage:

Purpose: Get the current code page of the client system.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_GetCodePage(  
    unsigned long *codePage,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

unsigned long * codePage - output

Returns the current code page of the client system or the OEM code page character conversion override value, if one is specified on the Language tab of the iSeries Access Properties dialog.

cwbSV_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved with the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: None

cwbNL_GetANSIcodePage:

Purpose: Get the current ANSI code page of the client system.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_GetANSIcodePage(  
    unsigned long *codePage,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

unsigned long * codePage - output

Returns the current ANSI code page of the client system or the ANSI code page character conversion override value, if one is specified on the Language tab of the iSeries Access Properties dialog.

cwbSV_ErrHandle errorHandle - output

Handle to an error object. Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved with the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Invalid handle.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: None

cwbNL_GetHostCCSID:

Purpose: Returns the associated CCSID of a given host system or the managing system or the EBCDIC code page character conversion override value, if one is specified on the Language tab of the iSeries Access Properties dialog.

Syntax:

```
unsigned long CWB_ENTRY cwbNL_GetHostCCSID(  
    char * system,  
    unsigned long * CCSID );
```

Parameters:

char * system - input

The name of the host system. If NULL, the managing system is used.

unsigned * CCSID - output

Length of the result buffer.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWBNL_DEFAULT_HOST_CCSID_USED

Host CCSID 500 is returned

Usage: This API does not make or require an active connection to the host system to retrieve the associated CCSID value. However, it does depend on a prior successful connection to the host system. If no prior successful connection was made to the host system, the API determines the most appropriate associated host CCSID by using an internal mapping table.

iSeries Access for Windows dialog-box NLS API list: iSeries Access for Windows dialog-box NLS APIs are interfaces that are used to manipulate the translatable text within dialog boxes.

The following iSeries Access for Windows dialog-box NLS APIs allow applications to:

- Replace translatable text with a dialog box
- Expand dialog-box controls according to the text

`cwbNL_CalcControlGrowthXY`

`cwbNL_CalcDialogGrowthXY`

`cwbNL_GrowControlXY`

`cwbNL_GrowDialogXY`

`cwbNL_LoadDialogStrings`

`cwbNL_LoadMenu`

`cwbNL_LoadMenuStrings`

`cwbNL_SizeDialog`

Usage notes

This module works **ONLY** on the following kinds of dialog-box controls:

- Static text
- Button
- Group box
- Edit box
- Check box
- Radio button

It does **NOT** work on complex controls such as Combo box.

cwbNL_CalcControlGrowthXY:

Purpose: Routine to calculate the growth factor of an individual control within a dialog box.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_CalcControlGrowthXY(  
    HWND windowHandle,  
    HDC hDC,  
    float* growthFactorX,  
    float* growthFactorY);
```

Parameters:

HWND windowHandle - input

Window handle of the control for which to calculate the growth factor.

HDC hDC - input

Device context. Used by **GetTextExtentPoint32** to determine extent needed for the translated string in the control.

float* growthFactorX - output

+/- growth to the width needed to contain the string for the control.

float* growthFactorY - output

+/- growth to the height needed to contain the string for the control.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion

Usage: It is assumed that the translated text has been loaded into the control prior to calling this function. A control that does not contain text will return a 1.00 growth factor. This means that it does not need to change size.

cwbNL_CalcDialogGrowthXY:

Purpose: Routine to calculate the growth factor of a dialog box. All of the controls within the dialog box will be looked at to determine how much the dialog-box size needs to be adjusted.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_CalcDialogGrowthXY(  
    HWND windowHandle,  
    float* growthFactorX,  
    float* growthFactorY);
```

Parameters:

HWND windowHandle - input

Window handle of the dialog box for which to calculate the growth factor.

float* growthFactorX - output

+/- growth to the width needed to contain the string for all of the controls in the dialog box.

float* growthFactorY - output

+/- growth to the height needed to contain the string for all of the controls in the dialog box.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion

Usage: It is assumed that the translated text has been loaded into the controls prior to calling this function.

cwbNL_GrowControlXY:

Purpose: Routine to grow an individual control within a dialog box.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_GrowControlXY(  
    HWND        windowHandle,  
    HWND        parentWindowHandle,  
    float        growthFactorX,  
    float        growthFactorY,  
    cwb_Boolean growAllControls);
```

Parameters:

HWND windowHandle - input

Window handle of the control to be resized.

HWND parentWindowHandle - input

Window handle of the dialog box that contains the controls.

float growthFactorX - input

Multiplication factor for growing the width of the control. 1.00 = Stay same size. 1.50 = 1 1/2 times original size.

float growthFactorY - input

Multiplication factor for growing the height of the control. 1.00 = Stay same size. 1.50 = 1 1/2 times original size.

cwb_Boolean growAllControls - input

CWB_TRUE = All controls will be resized by the growthFactor. CWB_FALSE = Only controls with text will be resized.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion

Usage: Care should be used to not pass in a growth factor that will cause a control to not fit on the physical display.

cwbNL_GrowDialogXY:

Purpose: Internal routine to growth the dialog box and its controls proportionally based off of a growth factor that is input.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_GrowDialogXY(  
    HWND        windowHandle,  
    float       growthFactorX,  
    float       growthFactorY,  
    cwb_Boolean growAllControls);
```

Parameters:

HWND windowHandle - input

Window handle of the window owning the controls.

float growthFactorX - input

Multiplication factor for growing the dialog box, ie. 1.00 = Stay same size, 1.50 = 1 1/2 times original size.

float growthFactorY - input

Multiplication factor for growing the dialog box, ie. 1.00 = Stay same size, 1.50 = 1 1/2 times original size.

cwb_Boolean growAllControls - input

CWB_TRUE = All controls will be resized by the growthFactor, CWB_FALSE = Only controls with text will be resized.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

Usage: It is assumed that the translated text has been loaded into the controls prior to calling this function. The dialog-box frame will not be allowed to grow larger than the desktop window size.

cwbnl_LoadDialogStrings:

Purpose: This routine will control the replacement of translatable text within a dialog box. This includes dialog control text as well as the dialog-box caption.

Syntax:

```
unsigned int CWB_ENTRY cwbnl_LoadDialogStrings(  
    HINSTANCE  MRIHandle,  
    HWND      windowHandle,  
    int       nCaptionID,  
    USHORT    menuID,  
    HINSTANCE  menuLibHandle,  
    cwb_Boolean growAllControls);
```

Parameters:

HINSTANCE MRIHandle - input

Handle of the module containing the strings for the dialog.

HWND windowHandle - input

Window handle of the dialog box.

int nCaptionID - input

ID of the caption string for the dialog box

USHORT menuID - input

ID of the menu for the dialog box.

HINSTANCE menuLibHandle - input

Handle of the module containing the menu for the dialog.

cwb_Boolean growAllControls - input

CWB_TRUE = All controls will be resized by the growthFactor CWB_FALSE = Only controls with text will be resized.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWBNL_DLG_MENU_LOAD_ERROR

Could not load the menu.

CWBNL_DLG_INVALID_HANDLE

Incorrect MRIHandle.

Usage: This process begins by enumerating, replacing the text of, and horizontally adjusting, all dialog controls within the dialog box, and finally right-adjusting the dialog box itself, relative to the adjusted controls therein. These adjustments are made only if the current window extents do not fully encompass the expansion space required for the text or all controls. After all of the text substitution has been completed, if a menu ID has been passed, it will be loaded and attached to the dialog box. It is suggested that this routine is called for every dialog-box procedure as the first thing done during the INITDLG message processing.

cwbnL_LoadMenu:

Purpose: This routine will control the loading of the given menu from a module and replacing the translatable text within the menu.

Syntax:

```
HWND CWB_ENTRY cwbnL_LoadMenu(  
    HWND    windowHandle,  
    HINSTANCE menuResourceHandle,  
    USHORT  menuID,  
    HINSTANCE MRIHandle);
```

Parameters:

HWND windowHandle - input

Window handle of the dialog box that contains the menu.

HINSTANCE menuResourceHandle - input

Handle of the resource dll containing the menu.

USHORT menuID - input

ID of the menu for the dialog box.

HINSTANCE MRIHandle - input

Handle of the resource dll containing the strings for the menu.

Return Codes: The following list shows common return values.

HINSTANCE

Handle of the menu.

Usage: None

cwbNL_LoadMenuStrings:

Purpose: This routine will control the replacement of translatable text within a menu.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_LoadMenuStrings(  
    HWND      WindowHandle,  
    HINSTANCE menuHandle,  
    HINSTANCE MRIHandle);
```

Parameters:

HWND windowHandle - input

Window handle of the dialog box that contains the menu.

HMODULE menuHandle - input

Handle of the menu for the dialog.

HMODULE MRIHandle - input

Handle of the resource DLL containing the strings for the menu.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion

Usage: None

cwbNL_SizeDialog:

Purpose: This routine will control the sizing of the dialog box and its child controls. The expansion amount is based off of the length of the text extent and the length of each control. The growth of the dialog box and its controls will be proportional. By setting the `growAllControls` to `FALSE`, only controls with text will expand or contract. This allows the programmer the flexibility of non-translatable fields to remain the same size. This may be appropriate for dialogs that contain drop-down lists, combo-boxes, or spin buttons.

Syntax:

```
unsigned int CWB_ENTRY cwbNL_SizeDialog(  
    HWND      windowHandle,  
    cwb_Boolean growAllControls);
```

Parameters:

HWND windowHandle - input

Window handle of the window owning the controls.

cwb_Boolean growAllControls - input

`CWB_TRUE` = All controls will be resized by the `growthFactor`, `CWB_FALSE` = Only controls with text will be resized.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion

Usage: This routine assumes that the translated text has already been loaded into the dialog-box controls. If the text has not been loaded into the controls, use **`cwbNL_LoadDialog`**.

Example: iSeries Access for Windows NLS APIs

```
/* National Language Support Code Snippet          */
/* Used to demonstrate how the APIs would be run. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "CWBNL.H"
#include "CWBNLCONV.H"
#include "CWBSV.H"

cwbSV_ErrHandle errhandle;

/* Return the message text associated with the top-level */
/* error identified by the error handle provided. Since */
/* all APIs that fail use the error handle, this was moved */
/* into a separate routine.                               */
void resolveErr(cwbSV_ErrHandle errhandle)
{
    static unsigned char buf[ BUFSIZ ];
    unsigned long retlen;
    unsigned int rc;

    if ((rc = cwbSV_GetErrText(errhandle, buf, (unsigned long) BUFSIZ, &retlen)) != CWB_OK)
        printf("cwbSV_GetErrText() Service API failed with return code 0x%x.\n", rc);
    else
        printf("%s\n", (char *) buf);
}

void main(void){

    /* define some variables
       ----- */
    int SVrc = 0;
    int NLrc = 0;
    char *myloadpath = "";
    char *resultPtr;
    char *mylang;
    unsigned short resultlen;
    unsigned short reqlen;
    unsigned long searchhandle;
    unsigned long codepage;
    unsigned long trgtpage;
    char *srcbuf = "Change this string";
    char *trgtbuf;
    unsigned long srclen;
    unsigned long trgtlen;
    unsigned long nmbrrrs;
    unsigned long posoferr;
    unsigned long rqdlen;
    unsigned long ccsid;

    /* Create an error message object and return a handle to */
    /* it. This error handle can be passed to APIs that */
    /* support it. If an error occurs, the error handle can */
    /* be used to retrieve the message text associated with */
    /* the API error.                                         */
    SVrc = cwbSV_CreateErrHandle(&errhandle);
    if (SVrc != CWB_OK) {
        printf("cwbSV_CreateErrHandle failed with return code %d.\n", SVrc);
    }

    /* Retrieve the current language setting.                */
    resultlen = CWBNL_MAX_LANG_SIZE+1;
    resultPtr = (char *) malloc(resultlen * sizeof(char));
    NLrc = cwbNL_GetLang(myloadpath, resultPtr, resultlen, &reqlen, errhandle);
    if (NLrc != CWB_NO_ERR) {
        if (NLrc == CWB_BUFFER_TOO_SMALL)
            printf("GetLang buffer too small, recommended size %d.\n", reqlen);
        resolveErr(errhandle);
    }
    printf("GetLang API returned %s.\n", resultPtr);
    mylang = (char *) malloc(resultlen * sizeof(char));
    strcpy(mylang, resultPtr);

    /* Retrieve the descriptive name of a language setting. */
    resultlen = CWBNL_MAX_NAME_SIZE+1;
    resultPtr = (char *) realloc(resultPtr, resultlen * sizeof(char));
    NLrc = cwbNL_GetLangName(mylang, resultPtr, resultlen, &reqlen, errhandle);
    if (NLrc != CWB_NO_ERR) {
```

```

    if (NLrc == CWB_BUFFER_TOO_SMALL)
        printf("GetLangName buffer too small, recommended size %d.\n", reqlen);
    resolveErr(errhandle);
}
printf("GetLangName API returned %s.\n", resultPtr);

/* Return the complete path for language files. */
resultlen = CWBNL_MAX_PATH_SIZE+1;
resultPtr = (char *) realloc(resultPtr, resultlen * sizeof(char));
NLrc = cwbNL_GetLangPath(myloadpath, resultPtr, resultlen, &reqlen, errhandle);
if (NLrc != CWB_NO_ERR) {
    if (NLrc == CWB_BUFFER_TOO_SMALL)
        printf("GetLangPath buffer too small, recommended size %d.\n", reqlen);
    resolveErr(errhandle);
}
printf("GetLangPath API returned %s.\n", resultPtr);

/* Get the code page of the current process. */
NLrc = cwbNL_GetCodePage(&codepage, errhandle);
if (NLrc != CWB_NO_ERR) {
    resolveErr(errhandle);
}
printf("GetCodePage API returned %u.\n", codepage);

/* Convert strings from one code page to another. This
/* API combines three converter APIs for the default
/* conversion. The three converter APIs it combines are:
/* cwbNL_CreateConverter
/* cwbNL_Convert
/* cwbNL_DeleteConverter
srcLen = strlen(srcbuf) + 1;
trgtlen = srcLen;
trgtpage = 437;
trgtbuf = (char *) malloc(trgtlen * sizeof(char));
printf("String to convert is %s.\n", srcbuf);
NLrc = cwbNL_ConvertCodePages(codepage, trgtpage, srcLen,
    trgtlen, srcbuf, trgtbuf, &nmbrrrs, &posoferr, &rqdlen,
    errhandle);
if (NLrc != CWB_NO_ERR) {
    resolveErr(errhandle);
    printf("number of errors detected is %u.\n", nmbrrrs);
    printf("location of first error is %u.\n", posoferr);
}
printf("ConvertCodePages API returned %s.\n", trgtbuf);

/* Map a code page to the corresponding CCSID. */
NLrc = cwbNL_CodePageToCCSID(codepage, &ccsid, errhandle);
if (NLrc != CWB_NO_ERR) {
    resolveErr(errhandle);
}
printf("CodePageToCCSID returned %u.\n", ccsid);

cwbSV_DeleteErrHandle(errhandle);
}

```

iSeries Access for Windows Directory Update APIs

What is iSeries Access for Windows Directory Update?

The iSeries Access for Windows Directory Update function allows users to specify PC directories for updating from a configured network server or from multiple networked servers. This permits users to load non-iSeries Access for Windows software products on a server in the network, and to keep those files updated on PCs. Directory Update is an optionally installable iSeries Access for Windows component.

How to install iSeries Access for Windows Directory Update:

To install Directory Update, follow these steps when you install iSeries Access for Windows, or when you run Selective Setup if iSeries Access for Windows is already installed:

1. Select the **iSeries Access for Windows Optional Components** check box.
2. Expand the view and make sure that the **Directory Update** subcomponent also is selected.
3. Follow the prompts to completion.

iSeries Access for Windows Directory Update C/C++ APIs:

iSeries Access for Windows Directory Update C/C++ application programming interfaces (APIs) allow software developers to add, change and delete update entries that are used by the iSeries Access for Windows Directory Update function.


Note: iSeries Access for Windows Directory Update APIs do not actually perform the updates. They are for configuration purposes only. The task of updating files is handled exclusively by the Directory Update application.

iSeries Access for Windows Directory Update APIs enable the:

- Creation of update entries.
- Deletion of update entries.
- Modification of update entries.
- Retrieval of information from update entries.
- Retrieval of information such as return codes. For example, only one application can access the Update entries at a time. If you get a return code that indicates **locked**, use the information to find the name of the application that has the entries open.

IMPORTANT: The iSeries Access for Windows client does not include support for network drives or for universal naming conventions. This now is provided by the **iSeries NetServer** function. Network drives that you previously mapped by using iSeries Access should be mapped by using iSeries NetServer support. Set up the iSeries NetServer that comes with OS/400 V4R2 and beyond in order to perform file serving to the iSeries server.

NetServer information resources:

- iSeries NetServer topic of the iSeries Information Center
- IBM iSeries NetServer Home Page 

iSeries Access for Windows Directory Update APIs required files:

Header file	Import library	Dynamic Link Library
cwbup.h	cwbapi.lib	cwbup.dll

Programmer's Toolkit:

The Programmer's Toolkit provides Directory Update documentation, access to the cwbup.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select **Directory Update** → **C/C++ APIs**.

iSeries Access for Windows Directory Update APIs topics:

- "Typical use of iSeries Access for Windows Directory Update APIs"
- "Requirements for Directory Update entries" on page 262
- "Options for Directory Update entries" on page 262
- "Directory Update package files syntax and format" on page 263
- **iSeries Access for Windows Directory Update APIs listing**
- "Directory Update sample program" on page 264
- "Directory Update APIs return codes" on page 27

Related topics:

- "iSeries system name formats for ODBC Connection APIs" on page 12
- "OEM, ANSI, and Unicode considerations" on page 12

Typical use of iSeries Access for Windows Directory Update APIs

iSeries Access for Windows Directory Update APIs typically are used for creating and configuring update entries that are used to update files from a mapped network drive. It is important to note that the Update APIs do not actually update the files, but rely on the Directory Update executable file to do this.

For example, files on the iSeries system might contain customer names and addresses. The files on your iSeries system are your master files that are updated as new customers are added, deleted, or have a name or address change. The same files on your networked personal computers are used to perform selective market mailings (by zip code, state, age, number of children and so on). The files on the iSeries system are your master files, and you want them secure, but you need to provide the data for work.

You could write a program that uses Directory Update APIs to create and configure update entries, which would update the files located on your networked personal computers.

Requirements for Directory Update entries

The following are required for Directory Update entries:

Description:

A description displayed by the Directory Update application to show users what is being updated.

Source path:

The path of the source or "master" files. For example:

E:\MYSOURCE

or

\\myserver\mysource

Target path:

The path of the files with which you wish to keep synchronized with the master files. For example:

C:\mytarget

Options for Directory Update entries

The following are optional for Directory Update entries:

Package files:

PC files that contain information on other files to be updated. See "Directory Update package files syntax and format" on page 263 for more information. Package files are added to update entries by using the "cwbUP_AddPackageFile" on page 265 API.

Callback DLL:

A DLL provided by the application programmer that Directory Update will call into during different stages of the update process. This allows programmers to perform application unique processing during the different stages of an update. A callback DLL is added to an update entry using the "cwbUP_SetCallbackDLL" on page 279 API.

The different stages of update when Directory Update may call into the callback DLL are:

Pre-update:

This is when Directory Update is about to begin its processing of an update entry. The following entry point prototype must be in the callback DLL: **unsigned long _declspec(dllexport) cwbUP_PreUpdateCallback();**

Post-update:

This is when Directory Update has completed moving the files. The following entry point prototype must be in the callback DLL: **unsigned long _declspec(dllexport) cwbUP_PostUpdateCallback();**

Pre-migration:

This is when Directory Update is about to begin version-to-version migration of an update entry. Version-to-version migrations are triggered by QPTFIDX files. The following entry point prototype must be in the callback DLL: **unsigned long _declspec(dllexport) cwbUP_PreMigrationCallback();**

Post-migration:

This is when Directory Update has completed processing of a version-to-version migration of an update entry. The following entry point prototype must be in the callback DLL:

```
unsigned long _cdeclspec(dllexport) cwUP_PostMigrationCallback();
```

Attributes:

Set the type or mode of the update to be performed. Combinations of the attributes are allowed. Attributes are:

File-driven update:

The files in the target directory are compared to the files in the source directory. Target files with dates older than the source files are updated. No new files will be created in the target.

Package-driven update:

The package files listed in the update entry are scanned for files to be updated. The dates of the files that are listed in the package file are compared between the source and the target directories. The source files with newer dates are updated or moved into the target directory. If a file that is listed in the package file does not exist in the target, but exists in the source, the file is created in the target directory.

Subdirectory update:

Subdirectories under the target directory are included in the update.

Onepass update:

Updates occur directly from source to target. If this is not specified, updates occur in two passes. The first pass of the update will copy the files to be updated into a temporary directory. Then the PC is restarted. On restart, the files are copied to the target directory. This is useful for locked files.

Backlevel update:

This controls if updates will occur if the source files are older than the target files.

Directory Update package files syntax and format

Package files contain information that specifies and describes which target files users want to be kept current with source files.

Package files syntax:

```
PKGF Description text
MBRF PROG1.EXE
MBRF INFO.TXT
MBRF SUBDIR\SHEET.XLS
DLTF PROG2.EXE
```

Note: Text must start in the first row and column of the file. Each package file must begin with the PKGF keyword.

Package files format:

Package files consist of the following elements:

PKGF description (optional):

This identifier indicates that the file is a package file. If this tag is not found in the first four characters of the file, Directory Update will not process the file while searching for files to update. A description is optional.

MBRF filename:

This identifies a file as part of the package to be updated. A path name also can be specified; this indicates that the file is in a subdirectory of the source directory.

The path should not contain the drive letter, or begin with a back-slash character (\). When you begin the update function, you specify a target directory; the path that is specified in the package file is considered a subdirectory of this target directory.


DLTF filename:

This identifies a file to be deleted from the target directory. A path name also can be specified; this indicates that the file is in a subdirectory of the target directory. As with the MBRF identifier, you should not specify a drive letter or begin with a back-slash character (\).

Related topic:

See “Directory Update sample program” for sample Directory Update APIs and detailed explanations of their attributes.

Directory Update sample program

A **Directory Update C/C++ sample program** is available when you link to the Programmer’s Toolkit – Directory Update Web page . Select **dirupdat.exe** for a description of the sample, and to download the samples.

The sample program demonstrates creating, configuring, and deleting Directory Update entries.

See the iSeries Access for Windows User’s Guide for more information.

iSeries Access for Windows Directory Update API listing

Note: It is essential that “cwbUP_FreeLock” on page 270 is called when your application no longer is accessing the update entries. If **cwbUP_FreeLock** is not called, other applications will not be able to access or modify the update entries.

The following iSeries Access for Windows Directory Update APIs are listed alphabetically, and are grouped by function:

Function	iSeries Access for Windows Directory Update APIs
Create an update entry	cwbUP_CreateUpdateEntry
Delete an update entry	cwbUP_DeleteEntry
Obtain access to an update entry	cwbUP_FindEntry cwbUP_FreeLock cwbUP_GetEntryHandle
Free resources that are associated with an entry handle	cwbUP_FreeEntryHandle
Change an update entry	cwbUP_AddPackageFile cwbUP_RemovePackageFile cwbUP_SetCallbackDLL cwbUP_SetDescription cwbUP_SetEntryAttributes cwbUP_SetSourcePath cwbUP_SetTargetPath
Obtain information from an update entry	cwbUP_GetCallbackDLL cwbUP_GetDescription cwbUP_GetEntryAttributes cwbUP_GetSourcePath cwbUP_GetTargetPath
Retrieve general Directory Update information	cwbUP_GetLockHolderName

cwbUP_AddPackageFile

Purpose: Adds a package file to the package file list in the update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_AddPackageFile(  
    cwbUP_EntryHandle entryHandle,  
    char *entryPackage);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or **cwbUP_FindEntry**.

char * entryPackage - input

Pointer to a null-terminated string that contains the name of a package file to be added to the update entry. Do not include the path for this file. The package file must exist in the source and target paths.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL was passed as an address.

CWBUP_TOO_MANY_PACKAGES

Maximum number of package files already exist for this entry.

CWBUP_STRING_TOO_LONG

The package file name is longer than CWBUP_MAX_LENGTH.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: None

cwbUP_CreateUpdateEntry

Purpose: Creates a new update entry and passes back a handle to it.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_CreateUpdateEntry(  
    char * entryDescription,  
    char * entrySource,  
    char * entryTarget,  
    cwbUP_EntryHandle *entryHandle);
```

Parameters:

char * entryDescription - input

Points to a null-terminated string that contains a description to identify the update entry.

char * entrySource - input

Points to a null-terminated string that contains the source for the update entry. This can be either a drive and path, or a UNC name.

char * entryTarget - input

Points to a null-terminated strings that contains the target for the update entry. This can be either a drive and path, or a UNC name.

cwbUP_EntryHandle * entryHandle - input/output

Pointer to a **cwbUP_EntryHandle** where the handle will be returned. This handle must be used in subsequent calls to the update entry APIs.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed as an address.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

CWBUP_TOO_MANY_ENTRIES

The maximum number of update entries already exist. No more can be created.

CWBUP_STRING_TOO_LONG

An input string is longer than the maximum of CWBUP_MAX_LENGTH.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: When you use this call, and have completed your processing of the update entry, you must call **cwbUP_FreeEntryHandle**. This call will "unlock" the entry, and free resources that are associated with it.

cwbUP_DeleteEntry

Purpose: Deletes the update entry from the update entry list.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_DeleteEntry(  
    cwbUP_EntryHandle entryHandle);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or **cwbUP_FindEntry**.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: After this call, you do not need to call **cwbUP_FreeEntryHandle**. The entry is "freed" when the entry is successfully deleted. If you retrieved the first update entry by using the **cwbUP_GetEntryHandle** API, and then called this API to delete the entry, all of the update entries would shift one position to fill the slot left by the delete. So, if you then wanted to get the next update item, you would pass the same index that you did on the previous **cwbUP_GetEntryHandle** API call.

cwbUP_FindEntry

Purpose: Gets a handle to an existing update entry by using entrySource and entryTarget as the search parameters.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_FindEntry(  
    char * entrySource,  
    char * entryTarget,  
    unsigned long *searchStart,  
    cwbUP_EntryHandle *entryHandle);
```

Parameters:

char * entrySource - input

Points to a null-terminated string that contains the source for the update entry. This can be either a drive and path, or a UNC name. This string will be used to search for a */ matching update entry.

char * entryTarget - input

Points to a null-terminated string that contains the target for the update entry. This can be either a drive and path, or a UNC name. This string will be used to search for a matching update entry.

unsigned long * searchStart - input/output

Pointer to an index into the list of update entries to begin the search at. This would be used in cases where multiple update entries may have matching source and targets. You would use this parameter to "skip" over entries in the search, and continue on searching for a matching update entry that is after searchStart in the list. On successful return, searchStart will be set to the position in the list where the update entry was found. This should be set to CWBUP_SEARCH_FROM_BEGINNING if you want to search all update entries.

cwbUP_EntryHandle * entryHandle - input/output

Pointer to a **cwbUP_EntryHandle** where the handle will be returned. This handle must be used in subsequent calls to the update entry APIs.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed as an address.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

CWBUP_SEARCH_POSITION_ERROR

Search starting position is not valid.

CWBUP_ENTRY_NOT_FOUND

No update entry matched search value.

CWBUP_STRING_TOO_LONG

An input string is longer than the maximum of CWBUP_MAX_LENGTH.

Usage: The handle that is returned from this call will be used for accessing the update entry with other Update APIs. When you use this call, and have completed your processing of the update entry, you must call **cwbUP_FreeEntryHandle**. This call will "unlock" the entry, and free resources with which it is associated.

cwbUP_FreeEntryHandle

Purpose: Frees an entry handle and all resources with which is associated.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_FreeEntryHandle(  
    cwbUP_EntryHandle entryHandle);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

The entry handle that is to be freed.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid or has already been

Usage: After this call you can no longer access the update entry. To access the update entry or another update entry, you would need to get a new entry handle.

cwbUP_FreeLock

Purpose: Frees the lock to the update entries. This should be called when the application is done accessing the update entries. If this is not called, other applications will not be able to access the update entries.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_FreeLock();
```

Parameters:

None

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBUP_UNLOCK_WARNING

Application did not have the update entries locked.

Usage: A lock to the update entries is obtained whenever an application accesses or changes an update entry. When the application no longer needs to access the update entries, the application should call this API.

cwbUP_GetCallbackDLL

Purpose: Gets the fully qualified name of the callback DLL for an update entry.

Syntax:

```
unsigned int CWB_ENTRY  cwbUP_GetCallbackDLL(  
                        cwbUP_EntryHandle entryHandle,  
                        char *dllPath,  
                        unsigned long bufferSize,  
                        unsigned long *actualLength);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * dllPath - input/output

Pointer to a buffer that will receive the fully qualified name of the DLL that will be called when individual stages of the update occur.

unsigned long bufferSize - input

Length of the dllPath buffer. Space should be included for the null termination character. If the buffer is not large enough to hold the entire DLL name, an error will be returned and the actualLength parameter will be set to the number of bytes the dllPath buffer needs to be.

unsigned long * actualLength - input/output

Pointer to a length variable that will be set to the size of the buffer needed to contain the fully qualified DLL name.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWB_BUFFER_OVERFLOW

Buffer is too small to hold return data.

Usage: None

cwbUP_GetDescription

Purpose: Gets the description of the update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_GetDescription(  
    cwbUP_EntryHandle entryHandle,  
    char *entryDescription,  
    unsigned long bufferLength,  
    unsigned long *actualLength);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * entryDescription - input/output

Pointer to a buffer that will receive the description of the update entry.

unsigned long bufferLength - input

Length of the buffer. An extra byte should be included for the null termination character. If the buffer is not large enough to hold the entire description, an error will be returned and the **actualLength** parameter will be set to the number of bytes the **entryDescription** buffer needs to be to contain the data.

unsigned long * actualLength - input/output

Pointer to a length variable that will be set to the size of the buffer needed to contain the description.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWB_BUFFER_OVERFLOW

Buffer is too small to hold return data.

Usage: None

cwbUP_GetEntryAttributes

Purpose: Gets the attributes of the update entry. These include: one pass update, file driven update, package driven update, and update subdirectories. Any combination of these is valid.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_GetEntryAttributes(  
    cwbUP_EntryHandle entryHandle,  
    unsigned long *entryAttributes);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

unsigned long * entryAttributes - input/output

Pointer to area to receive the attribute values. (See defines section for values)

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

Usage: The value that is contained in entryAttributes after this call is made may be a combination of the attribute flags that are listed near the top of this file.

cwbUP_GetEntryHandle

Purpose: Gets a handle to an existing update entry at a given position in the list.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_GetEntryHandle(  
    unsigned long entryPosition,  
    cwbUP_EntryHandle *entryHandle);
```

Parameters:

unsigned long entryPosition - input

Index into the update entry list of the entry for which you want to retrieve a handle. (Pass in 1 if you wish to retrieve the first update entry)

cwbUP_EntryHandle * entryHandle - input/output

Pointer to a **cwbUP_EntryHandle** where the handle will be returned. This handle must be used in subsequent calls to the update entry APIs.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL was passed as an address.

CWBUP_ENTRY_NOT_FOUND

No update entry at the given position.

CWBUP_POSITION_INVALID

Position that is given is not in range.

Usage: The handle that is returned from this call will be used for accessing the update entry with other Update APIs. When you use this call, and have completed your processing of the update entry, you must call **cwbUP_FreeEntryHandle**. This call will "unlock" the entry, and free resources that are associated with it. You must call **cwbUP_FreeEntryHandle** once for each time that you call an API that returns an entry handle.

cwbUP_GetLockHolderName

Purpose: Gets the name of the program that currently has the update entries in a locked state.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_GetLockHolderName(char *lockHolder,  
                                              unsigned long bufferLength,  
                                              unsigned long *actualLength);
```

Parameters:

char * lockHolder - input/output

Pointer to a buffer that will receive the name of the application that is currently locking the update entries.

unsigned long bufferLength - input

Length of the buffer. An extra byte should be included for the null termination character. If the buffer is not large enough to hold the entire name, an error will be returned and the actualLength parameter will be set to the number of bytes the lockHolder buffer needs to be to contain the data.

unsigned long * actualLength - input/output

Pointer to a length variable that will be set to the size of the buffer needed to contain the application name.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWB_BUFFER_OVERFLOW

Buffer is too small to hold return data.

Usage: None

cwbUP_GetSourcePath

Purpose: Gets the source path of the update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_GetSourcePath(  
    cwbUP_EntryHandle entryHandle,  
    char *entrySource,  
    unsigned long bufferLength,  
    unsigned long *actualLength);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * entrySource - input/output

Pointer to a buffer that will receive the source path of the update entry.

unsigned long bufferLength - input

Length of the buffer. An extra byte should be included for the null termination character. If the buffer is not large enough to hold the entire source path, an error will be returned and the **actualLength** parameter will be set to the number of bytes the **entrySource** buffer needs to be to contain the data.

unsigned long * actualLength - input/output

Pointer to a length variable that will be set to the size of the buffer needed to contain the source path.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWB_BUFFER_OVERFLOW

Buffer is too small to hold return data.

Usage: None

cwbUP_GetTargetPath

Purpose: Gets the target path of the update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_GetTargetPath(  
    cwbUP_EntryHandle entryHandle,  
    char *entryTarget,  
    unsigned long bufferSize,  
    unsigned long *actualLength);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * entryTarget - input/output

Pointer to a buffer that will receive the target path of the update entry.

unsigned long bufferSize - input

Length of the buffer. An extra byte should be included for the null termination character. If the buffer is not large enough to hold the entire target path, an error will be returned and the **actualLength** parameter will be set to the number of bytes the **entryTarget** buffer needs to be to contain the data.

unsigned long * actualLength - input/output

Pointer to a length variable that will be set to the size of the buffer needed to contain the target path.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWB_BUFFER_OVERFLOW

Buffer is too small to hold return data.

Usage: None

cwbUP_RemovePackageFile

Purpose: Removes a package file from the list of package files that belong to an update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_RemovePackageFile(  
    cwbUP_EntryHandle entryHandle,  
    char *entryPackage);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * entryPackage - input

Pointer to a null-terminated string that contains the package file name that is to be removed from the package file list.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWBUP_PACKAGE_NOT_FOUND

The package file was not found.

CWBUP_STRING_TOO_LONG

The package file string is longer than the maximum of CWBUP_MAX_LENGTH.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: None

cwbUP_SetCallbackDLL

Purpose: Sets the fully qualified name of the callback DLL for an update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_SetCallbackDLL(  
    cwbUP_EntryHandle entryHandle,  
    char *dllPath);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or **cwbUP_FindEntry**.

char * dllPath - input

Pointer to a null-terminated string that contains the fully qualified name of the DLL that will be called when individual stages of the update occur.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWBUP_STRING_TOO_LONG

The callback DLL string is longer than the maximum of **CWBUP_MAX_LENGTH**.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: None

cwbUP_SetDescription

Purpose: Sets the description of the update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_SetDescription(  
    cwbUP_EntryHandle entryHandle,  
    char *entryDescription);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * entryDescription - input

Pointer to a null-terminated string that contains the full description to be associated with the update entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWBUP_STRING_TOO_LONG

The description string is longer than the maximum of **CWBUP_MAX_LENGTH**.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: None

cwbUP_SetEntryAttributes

Purpose: Sets any of the following attribute values of the update entry:

CWBUP_FILE_DRIVEN

Updates are based on file date comparisons between target and source files.

CWBUP_PACKAGE_DRIVEN

Updates are based on contents of the package file(s), and comparisons of their files' dates between target and source.

CWBUP_SUBDIRECTORY

Update compares and updates directories under the given path.

CWBUP_ONEPASS

Updates occur directly in one pass. If this isn't specified, updates occur in two passes. The first pass copies the files to be updated to a temporary directory, and then when the PC is rebooted, the files are copied to the target directory.

CWBUP_BACKLEVEL_OK

If this is set, updates will occur if the dates of the files on the source and target don't match. If this is not set, updates will only occur if the source file is more recent than the target file.

Any combination of these values is valid.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_SetEntryAttributes(  
    cwbUP_EntryHandle entryHandle,  
    unsigned long entryAttributes);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

unsigned long entryAttributes - input

Combination of the attribute values. (See defines section for values)

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: An example of this call follows:

```
rc = cwbUP_SetEntryAttributes(entryHandle, CWBUP_FILEDRIVEN | CWBUP_ONEPASS );
```

This call would result in the update entry being file driven and the update would occur in one pass.

cwbUP_SetSourcePath

Purpose: Sets the source path of the update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_SetSourcePath(  
    cwbUP_EntryHandle entryHandle,  
    char *entrySource);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * entrySource - input

Pointer to a null-terminated string that contains the full source path for the update entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWBUP_STRING_TOO_LONG

The source path string is longer than the maximum of **CWBUP_MAX_LENGTH**.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: None

cwbUP_SetTargetPath

Purpose: Sets the target path of the update entry.

Syntax:

```
unsigned int CWB_ENTRY cwbUP_SetTargetPath(  
    cwbUP_EntryHandle entryHandle,  
    char *entryTarget);
```

Parameters:

cwbUP_EntryHandle entryHandle - input

Handle that was returned by a previous call to **cwbUP_CreateUpdateEntryHandle**, **cwbUP_GetUpdateEntryHandle**, or to **cwbUP_FindEntry**.

char * entryTarget - input

Pointer to a null-terminated string that contains the full target path for the update entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Update entry handle is not valid.

CWB_INVALID_POINTER

NULL passed as an address parameter.

CWBUP_STRING_TOO_LONG

The target path string is longer than the maximum of **CWBUP_MAX_LENGTH**.

CWBUP_ENTRY_IS_LOCKED

Another application is currently changing the update entry list. No changes are allowed at this time.

Usage: None

iSeries Access for Windows PC5250 emulation APIs

The iSeries Access for Windows PC5250 emulator provides desktop users with a graphical user interface for existing iSeries applications. PC5250 allows users to easily and transparently interact with data and applications that are stored on the iSeries server. PC5250 provides C/C++ application programming interfaces (APIs) for enabling workstation programs to interact with iSeries host systems.

iSeries Access for Windows PC5250 C/C++ APIs:

Emulator high-level language API (EHLLAPI)

A simple, single-entry point interface that interprets the emulator screen.

Personal communications session API (PCSAPI)

Use this interface to start, stop, and control emulator sessions.

Host Access Class Library (HACL)

This interface provides a set of classes and methods for developing applications that access host information at the data-stream level.

iSeries Access for Windows emulation APIs required files:

Emulation interface	Header file	Import library	Dynamic Link Library
Standard HLLAPI	hapi_c.h	pscal32.lib	pcshll.dll pcshll32.dll
Enhanced HLLAPI	ehlapi32.h	ehlapi32.lib	ehlapi32.dll
Windows EHLLAPI	whllapi.h	whllapi.lib whllapi32.lib	whllapi.dll whllapi32.dll
HACL interface	eclall.hpp	pcseclva.lib pcseclvc.lib	pcseclva.dll pcseclvc.dll
PCSAPI interface	pcsapi.h	pscal32.lib	pcsapi.dll pcsapi32.dll

Programmer's Toolkit:

The Programmer's Toolkit provides Emulator interfaces documentation, access to header files, and links to sample applications. To access this information, open the Programmer's Toolkit and select **Emulation** → **C/C++ APIs**.

IBM Lightweight Directory Access Protocol (LDAP) APIs

LDAP is a protocol for accessing online directory services directly over TCP/IP. It provides TCP/IP access to X.500 directories, as well as other stand-alone LDAP servers. LDAP APIs are used to provide both synchronous and asynchronous access to a directory.

LDAP is a global directory service that is based on a client/server model. One or more LDAP servers contain data that form the LDAP directory tree. An LDAP client connects to an LDAP server, and makes a request for information. The server responds with an answer or with a pointer to where the client can obtain more information. Typically, this is another LDAP server. Regardless of the LDAP server, the client is always presented with the same view of the directory: A name presented to one LDAP server references the identical entry that would be referenced at another LDAP server.

The SecureWay Directory Client SDK (Version 3.1.1 for Windows 98/NT/2000/XP) is a Software Developer's Kit that provides development support for LDAP applications. The latest version of SDK can be found at the IBM SecureWay Directory home page at <http://www.ibm.com/software/network/directory>. It can be installed by running the `/qibm/proddata/os400/dirsrv/usertools/windows/setup.exe` program. This program is installed with OS/400 installation option 3 in V5R2.

You can map a drive to the iSeries server using one of the following predefined share names:

- QIBM
- /qibm QDIRSRV
- /qibm/proddata/os400/dirsrv

For example, `\\myas400\qdirsrv\usertools\windows\setup`, will map the SDK using the QDIRSRV share name. This example uses an image for the InstallShield Package For The Web. When executed, this example will extract the files to a temporary directory on the PC and run the installation program. The user is prompted for the language to install (English by default), the install path (c:\program files\ibm\ldap by default), and the program folder (IBM SecureWay Directory). On completion, the temporary files are deleted.

LDAP C APIs are no longer included with iSeries Access for Windows, however, these APIs are part of the IBM SecureWay Directory Client SDK for Windows 98/NT/2000/XP. The IBM SecureWay Directory Client SDK consists of:

- C header files, library files, on-line documentation and sample programs
- Java Naming and Directory Interface (JNDI) LDAP service provider (IBMJNDI)
- LDAP command line utilities
- Directory Management Tool (DMT), a directory content management graphical user interface.

For further information on the IBM SecureWay Directory, see the IBM SecureWay Directory (www.ibm.com/software/network/directory) home page. For further information on OS/400 Directory Services, see the home page for iSeries Directory Services (<http://www.ibm.com/eserver/series/ldap>).

iSeries Access for Windows Multimedia APIs

Ultimedia System Facilities (USF) is an object-based multimedia management system, and an integrated function of OS/400. It provides a series of functions that are available to applications through application program interfaces (APIs). USF APIs are available to both iSeries server and client PC applications. These APIs use standard interfaces that are callable from high-level languages (COBOL, RPG, and C). The client APIs are callable from programs that support C-style APIs. The API requests are routed to either the iSeries server or a client PC, depending on which platform is best suited to perform the function.

Support for Ultimedia System Facilities APIs resides in the application layer above the operating system. It uses standard operating system interfaces, including:

- iSeries Access for communications between the iSeries system and the client
- Network drives to store both byte-stream multimedia data and attribute data
- Multimedia extensions that are supplied by Microsoft Windows environments
- Standard graphical user interface-support of the Windows environment
- iSeries security for the protection of objects that are stored in the Ultimedia System Facilities Multimedia Repository

Programmer's Toolkit:

The Programmer's Toolkit provides USF APIs documentation, access to the USF interface definition (header) files, and links to sample programs. To access this information, open the Programmer's Toolkit and select **Multimedia** → **C/C++ APIs**.

iSeries Access for Windows Multimedia topics:

- "Ultimedia System Facilities API capabilities overview"
- "Ultimedia System Facilities API types overview" on page 286

Ultimedia System Facilities API capabilities overview

Ultimedia System Facilities APIs allow you to do the following:

- Add multimedia functions to existing iSeries applications
- Add multimedia interfaces to existing iSeries applications with little modification to the existing application

- Create new iSeries applications that use multimedia
- Create new client applications that use multimedia
- Create cooperative (iSeries and client) applications that use multimedia
- Use the iSeries system as a Multimedia Repository and server for either iSeries-based or client-based tools or applications
- Use diverse multimedia devices such as digital video adapters, compact disc (CD) players, videodisc players, videocassette recorders (VCRs), and audio boards through the Multimedia Extensions Media Control Interface
- Share video media without physically handling the video devices or having players attached to dedicated viewing stations
- Sequence the delivery and presentation of multiple multimedia objects to the client
- Import and track multimedia objects that were created by industry-standard authoring tools
- Use multimedia input and output such as touch screens, audio output, full-motion video, and images

Ultimedia System Facilities API types overview

The following types of APIs comprise the Ultimedia System Facilities APIs:

Object Management APIs

These APIs query, create, change, and delete Ultimedia System Facilities objects and their attributes.

Multimedia APIs

These APIs support various ways of capturing, editing, and presenting multimedia objects. They also allow the integration of multimedia into an iSeries server or client application.

Shared Analog Device Control (SADC) APIs

These APIs control the shared multimedia analog devices that are attached to the iSeries system.

Cooperative Process Management (CPM) APIs

Every client Ultimedia System Facilities application must use the two CPM APIs that start and stop CPM processing (fzzmInitializeUSFComm and fzzmStopUSFComm). Additional CPM APIs provide optional functions such as:

- Sending a request for processing from one platform to the other
- Returning request data
- Retrieving data sent from another process

iSeries Objects APIs for iSeries Access for Windows

iSeries Objects for iSeries Access for Windows application programming interfaces (APIs) allow you to work with iSeries print-related objects. These APIs make it possible to work with iSeries spooled files, writer jobs, output queues, printers, and more.

By using iSeries Objects APIs, you can write workstation applications that are customized for the user's environment. For example, you can write an application to manage spooled files for a single user, or for all users across a network of iSeries servers. This includes holding, releasing, changing attributes of, deleting, sending, retrieving and answering messages for the spooled files.

iSeries Objects APIs for iSeries Access for Windows required files:

Header file	Import library	Dynamic Link Library
cwbobj.h	cwbapi.lib	cwbobj.dll

Programmer's Toolkit:

The Programmer's Toolkit provides iSeries Objects documentation, access to the cwbobj.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select **iSeries Operations** —> **C/C++ APIs**.

iSeries Objects APIs for iSeries Access for Windows topics:

- “iSeries objects attributes”
- **iSeries Objects API for iSeries Access for Windows listing**
- “Example: Using iSeries Objects APIs for iSeries Access for Windows” on page 397
- “iSeries Object APIs return codes” on page 28

Related topics:

- “iSeries system name formats for ODBC Connection APIs” on page 12
- “OEM, ANSI, and Unicode considerations” on page 12

iSeries objects attributes

Network Print Server objects have attributes. The Network Print Server supports the following attributes. Refer to the data stream description for each object/action to determine the attributes that are supported for that combination.

iSeries objects attributes listing

"Advanced Function Printing" on page 287	"Front Overlay Offset Across" on page 297	"Unprintable Characters" on page 306
"Align Page" on page 289	"Front Overlay Offset Down" on page 297	"Separation Character" on page 306
"Allow Direct Print" on page 289	"Graphic Character Set" on page 297	"Resource library name" on page 306
"Authority" on page 289	"Hardware Justification" on page 297	"Resource name" on page 306
"Authority to Check" on page 289	"Hold Spool File" on page 297	"Resource object type" on page 306
"Automatically End Writer" on page 298	"Job Name" on page 298	"Restart Printing" on page 307
"Back Margin Offset Across" on page 298	"Job Number" on page 298	"Save Spooled File" on page 307
"Back Margin Offset Down" on page 298	"Job Name" on page 298	"Seek Offset" on page 307
"Backside Overlay Library Name" on page 298	"Job Number" on page 298	"Seek Origin" on page 307
"Backside Overlay Name" on page 298	"Job Separators" on page 298	"Send Priority" on page 307
"Back Overlay offset across" on page 298	"Job Name" on page 298	"Separator page" on page 307
"Back Overlay Offset Down" on page 298	"Job Page Printed" on page 299	"Source Drawer" on page 308
"Characters per Inch" on page 299	"Length of Page" on page 299	"Spool SCS" on page 308
"Code Page" on page 291	"Library Name" on page 299	"Spool the Data" on page 308
"Coded Font Name" on page 291	"Lines Per Inch" on page 299	"Spooled File Name" on page 308
"Coded Font Library Name" on page 291	"Manufacturer Type and Model" on page 299	"Spooled File Number" on page 308
"Copies" on page 291	"Maximum Spooled Output Record Spooled File Size" on page 309	"Spooled File Status" on page 308
"Copies left to Produce" on page 291	"Measurement Method" on page 300	"Spooled Output Schedule" on page 309
"Current Page" on page 291	"Message Help" on page 300	"Starting Page" on page 309
"Data Format" on page 292	"Message ID" on page 300	"Text Description" on page 309
"Data Queue Library Name" on page 292	"Message Queue Library Name" on page 300	"Time File Opened" on page 309
"Data Queue Name" on page 292	"Message Queue" on page 300	"Total Pages" on page 309
"Date File Opened" on page 292	"Message Reply" on page 300	"Transform SCS to ASCII" on page 309
"User Specified DBCS Data" on page 292	"Message Text" on page 301	"Unit of Measure" on page 310
"DBCS Extension Characters" on page 292	"Message Type" on page 301	"User Comment" on page 310
"DBCS Character Rotation" on page 292	"Message Severity" on page 301	"User Data" on page 310
"DBCS Characters per Inch" on page 292	"Number of Bytes to Read/Write" on page 301	"User defined data" on page 310
"DBCS SO/SI Spacing" on page 292	"Number of Files" on page 301	"User defined object library" on page 310
"Defer Write" on page 293	"Number of Writers Started to Queue" on page 302	"User defined object name" on page 310
"Degree of Page Rotation" on page 293	"Number of Extended Attribute" on page 302	"User defined object type" on page 311
"Delete File After Sending" on page 294	"Operator commands" on page 302	"User defined option(s)" on page 311
"Destination Option" on page 294	"Operator Controlled" on page 302	"User driver program" on page 311
"Destination Type" on page 294	"Order of Files On Queue" on page 302	"User driver program library" on page 311
"Device Class" on page 294	"Output Priority" on page 302	"User driver program name" on page 311
"Device Model" on page 294	"Output Queue Library Name" on page 303	"User ID Address" on page 312
"Device Type" on page 294	"Output Queue Name" on page 303	"User transform program library" on page 312
"Display any File" on page 294	"Output Queue Status" on page 303	"User transform program name" on page 312
"Drawer for Separators" on page 295	"Overflow Line Number" on page 303	"VM/MVS Class" on page 312
"Ending Page" on page 295	"Pages Per Side" on page 303	"When to Automatically End Writer" on page 312
"File Separators" on page 295	"Pel Density" on page 303	"When to End Writer" on page 312
"Fold Records" on page 295	"Point Size" on page 304	"When to Hold File" on page 313
"Font Identifier" on page 295	"Print Fidelity" on page 304	"Width of Page" on page 313
"Form Feed" on page 296	"Print on Both Sides" on page 304	"Workstation Customizing Object Name" on page 313
"Form Type" on page 296	"Print Quality" on page 304	"Workstation Customizing Object Library" on page 313
"Form Type Message Option" on page 296	"Print Sequence" on page 304	"Writer Job Name" on page 313
"Front Margin Offset Across" on page 296	"Print Text" on page 304	"Writer Job Number" on page 313
"Front Margin Offset Down" on page 296	"Printer Device Type" on page 305	"Writer Job Status" on page 314
"Front Overlay Library Name" on page 297	"Printer File Library Name" on page 305	"Writer Job User Name" on page 314
"Front Overlay Name" on page 297	"Printer File Name" on page 305	"VM/SPS Starting Page" on page 314
	"Printer Queue" on page 305	"Network Print Server Object Attributes" on page 314
	"Record Length" on page 305	
	"Remote System" on page 306	

Advanced Function Printing

Key CWBOBJ_KEY_AFP

ID 0x000A

Type char[11]

Description

Indicates whether this spooled file uses AFP resources external to the spooled file. Valid values are *YES and *NO.

Align Page

Key CWBOBJ_KEY_ALIGN

ID 0x000B

Type char[11]

Description

Indicates whether a forms alignment message is sent prior to printing this spooled file. Valid values are *YES, *NO.

Allow Direct Print

Key CWBOBJ_KEY_ALWDRTprt

ID 0x000C

Type char[11]

Description

Indicates whether the printer writer allows the printer to be allocated to a job that prints directly to a printer. Valid values are *YES, *NO.

Authority

Key CWBOBJ_KEY_AUT

ID 0x000D

Type char[11]

Description

Specifies the authority that is given to users who do not have specific authority to the output queue. Valid values are *USE, *ALL, *CHANGE, *EXCLUDE, *LIBCRTAUT.

Authority to Check

Key CWBOBJ_KEY_AUTCHK

ID 0x000E

Type char[11]

Description

Indicates what type of authorities to the output queue allow the user to control all the files on the output queue. Valid values are *OWNER, *DTAAUT.

Automatically End Writer

Key CWBOBJ_KEY_AUTOEND

ID 0x0010

Type char[11]

Description

Specifies if the writer should be automatically ended. Valid values are *NO, *YES.

Back Margin Offset Across

Key CWBOBJ_KEY_BACKMGN_ACR

ID 0x0011

Type float

Description

For the back side of a piece of paper, it specifies, how far in from the left side of the page printing starts. The special value *FRONTMGN will be encoded as -1.

Back Margin Offset Down

Key CWBOBJ_KEY_BACKMGN_DWN

ID 0x0012

Type float

Description

For the back side of a piece of paper, it specifies, how far down from the top of the page printing starts. The special value *FRONTMGN will be encoded as -1.

Backside Overlay Library Name

Key CWBOBJ_KEY_BKOVRLIB

ID 0x0013

Type char[11]

Description

The name of the library that contains the back overlay. If the back overlay name field has a special value, this library field will be blank.

Backside Overlay Name

Key CWBOBJ_KEY_BKOVRLAY

ID 0x0014

Type char[11]

Description

The name of the back overlay. Valid special values include *FRONTMGN.

Back Overlay offset across

Key CWBOBJ_KEY_BKOVL_ACR

ID 0x0016

Type float

Description

The offset across from the point of origin where the overlay is printed.

Back Overlay Offset Down

Key CWBOBJ_KEY_BKOVL_DWN

ID 0x0015

Type float

Description

The offset down from the point of origin where the overlay is printed.

Characters per Inch

Key CWBOBJ_KEY_CPI

ID 0x0017

Type float

Description

The number of characters per horizontal inch.

Code Page

Key CWBOBJ_KEY_CODEPAGE

ID 0x0019

Type char[11]

Description

The mapping of graphic characters to code points for this spooled file. If the graphic character set field contains a special value, this field may contain a zero (0).

Coded Font Name

Key CWBOBJ_KEY_CODEDFNT

ID 0x001A

Type char[11]

Description

The name of the coded font. A coded font is an AFP resource that is composed of a character set and a code page. Special values include *FNTCHRSET.

Coded Font Library Name

Key CWBOBJ_KEY_CODEDFNTLIB

ID 0x0018

Type char[11]

Description

The name of the library that contains the coded font. This field may contain blanks if the coded font name field has a special value.

Copies

Key CWBOBJ_KEY_COPIES

ID 0x001C

Type long

Description

The total number of copies to be produced for this spooled file.

Copies left to Produce

Key CWBOBJ_KEY_COPIESLEFT

ID 0x001D

Type long

Description

The remaining number of copies to be produced for this spooled file.

Current Page

Key CWBOBJ_KEY_CURPAGE

ID 0x001E

Type long

Description

Current page that is being written by the writer job.

Data Format

Key CWBOBJ_KEY_DATAFORMAT

ID 0x001F

Type char[11]

Description

Data format. Valid values are *RCDDATA, *ALLDATA.

Data Queue Library Name

Key CWBOBJ_KEY_DATAQUELIB

ID 0x0020

Type char[11]

Description

The name of the library that contains the data queue.

Data Queue Name

Key CWBOBJ_KEY_DATAQUE

ID 0x0021

Type char[11]

Description

Specifies the name of the data queue that is associated with the output queue.

Date File Opened

Key CWBOBJ_KEY_DATE

ID 0x0022

Type char[8]

Description

The date the spooled file was opened. The date is encoded in a character string with the following format, C YY MM DD.

User Specified DBCS Data

Key CWBOBJ_KEY_DBCSDATA

ID 0x0099

Type char[11]

Description

Whether the spooled file contains double-byte character set (DBCS) data. Valid values are *NO and *YES.

DBCS Extension Characters

Key CWBOBJ_KEY_DBCSEXTENSN

ID 0x009A

Type char[11]

Description

Whether the system is to process the DBCS extension characters. Valid values are *NO and *YES.

DBCS Character Rotation

Key CWBOBJ_KEY_DBCAROTATE

ID 0x009B

Type char[11]

Description

Whether the DBCS characters are rotated 90 degrees counterclockwise before printing. Valid values are *NO and *YES.

DBCS Characters per Inch

Key CWBOBJ_KEY_DBCSCPI

ID 0x009C

Type long

Description

The number of double-byte characters to be printed per inch. Valid values are -1, -2, 5, 6, and 10. The value *CPI is encoded as -1. The value *CONDENSED is encoded as -2.

DBCS SO/SI Spacing

Key CWBOBJ_KEY_DBCSSISO

ID 0x009D

Type char[11]

Description

Determines the presentation of shift-out and shift-in characters when printed. Valid values are *NO, *YES, and *RIGHT.

Defer Write

Key CWBOBJ_KEY_DFR_WRITE

ID 0x0023

Type char[11]

Description

Whether print data is held in system buffers before

Degree of Page Rotation

Key CWBOBJ_KEY_PAGRTT

ID 0x0024

Type long

Description

The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. Valid values are -1, -2, -3, 0, 90, 180, 270. The value *AUTO is encoded as -1, the value *DEVD is encoded as -2, and the value *COR is encoded as -3.

Delete File After Sending

Key CWBOBJ_KEY_DELETESPLF

ID 0x0097

Type char[11]

Description

Delete the spooled file after sending? Valid values are *NO and *YES.

Destination Option

Key CWBOBJ_KEY_DESTOPTION

ID 0x0098

Type char[129]

Description

Destination option. A text string that allows the user to pass options to the receiving system.

Destination Type

Key CWBOBJ_KEY_DESTINATION

ID 0x0025

Type char[11]

Description

Destination type. Valid values are *OTHER, *AS400, *PSF2.

Device Class

Key CWBOBJ_KEY_DEVCLASS

ID 0x0026

Type char[11]

Description

The device class.

Device Model

Key CWBOBJ_KEY_DEVMODEL

ID 0x0027

Type char[11]

Description

The model number of the device.

Device Type

Key CWBOBJ_KEY_DEVTYPE

ID 0x0028

Type char[11]

Description

The device type.

Display any File

Key CWBOBJ_KEY_DISPLAYANY

ID 0x0029

Type char[11]

Description

Whether users who have authority to read this output queue can display the output data of any output file on this queue, or only the data in their own files. Valid values are *YES, *NO, *OWNER.

Drawer for Separators

Key CWBOBJ_KEY_DRWRSEP

ID 0x002A

Type long

Description

Identifies the drawer from which the job and file separator pages are to be taken. Valid values are -1, -2, 1, 2, 3. The value *FILE is encoded as -1, and the value *DEV D is encoded as -2.

Ending Page

Key CWBOBJ_KEY_ENDPAGE

ID 0x002B

Type long

Description

The page number at which to end printing the spooled file. Valid values are 0 or the ending page number. The value *END is encoded as 0.

File Separators

Key CWBOBJ_KEY_FILESEP

ID 0x002C

Type long

Description

The number of file separator pages that are placed at the beginning of each copy of the spooled file. Valid values are -1, or the number of separators. The value *FILE is encoded as -1.

Fold Records

Key CWBOBJ_KEY_FOLDREC

ID 0x002D

Type char[11]

Description

Whether records that exceed the printer forms width are folded (wrapped) to the next line. Valid values are *YES, *NO.

Font Identifier

Key CWBOBJ_KEY_FONTID

ID 0x002E

Type char[11]

Description

The printer font that is used. Valid special values include *CPI and *DEV D.

Form Feed

Key CWBOBJ_KEY_FORMFEED

ID 0x002F

Type char[11]

Description

The manner in which forms feed to the printer. Valid values are *CONT, *CUT, *AUTOCUT, *DEVD.

Form Type

Key CWBOBJ_KEY_FORMTYPE

ID 0x0030

Type char[11]

Description

The type of form to be loaded in the printer to print this spooled file.

Form Type Message Option

Key CWBOBJ_KEY_FORMTYPEMSG

ID 0x0043

Type char[11]

Description

Message option for sending a message to the writer's message queue when the current form type is finished. Valid values are *MSG, *NOMSG, *INFOMSG, *INQMSG.

Front Margin Offset Across

Key CWBOBJ_KEY_FTMGN_ACR

ID 0x0031

Type float

Description

For the front side of a piece of paper, it specifies, how far in from the left side of the page printing starts. The special value *DEVD is encoded as -2.

Front Margin Offset Down

Key CWBOBJ_KEY_FTMGN_DWN

ID 0x0032

Type float

Description

For the front side of a piece of paper, it specifies, how far down from the top of the page printing starts. The special value *DEVD is encoded as -2.

Front Overlay Library Name

Key CWBOBJ_KEY_FTOVRLLIB

ID 0x0033

Type char[11]

Description

The name of the library that contains the front overlay. This field may be blank if the front overlay name field contains a special value.

Front Overlay Name

Key CWBOBJ_KEY_FTOVRLAY

ID 0x0034

Type char[11]

Description

The name of the front overlay. Valid special values include *NONE.

Front Overlay Offset Across

Key CWBOBJ_KEY_FTOVL_ACR

ID 0x0036

Type float

Description

The offset across from the point of origin where the overlay is printed.

Front Overlay Offset Down

Key CWBOBJ_KEY_FTOVL_DWN

ID 0x0035

Type float

Description

The offset down from the point of origin where the overlay is printed.

Graphic Character Set

Key CWBOBJ_KEY_CHAR_ID

ID 0x0037

Type char[11]

Description

The set of graphic characters to be used when printing this file. Valid special values include *DEVD, *SYSVAL, and *JOBCCSID.

Hardware Justification

Key CWBOBJ_KEY_JUSTIFY

ID 0x0038

Type long

Description

The percentage that the output is right justified. Valid values are 0, 50, 100.

Hold Spool File

Key CWBOBJ_KEY_HOLD

ID 0x0039

Type char[11]

Description

Whether the spooled file is held. Valid values are *YES, *NO.

Initialize the writer

Key CWBOBJ_KEY_WTRINIT

ID 0x00AC

Type char[11]

Description

The user can specify when to initialize the printer device. Valid values are *WTR, *FIRST, *ALL.

Internet Address

Key CWBOBJ_KEY_INTERNETADDR

ID 0x0094

Type char[16]

Description

The internet address of the receiving system.

Job Name

Key CWBOBJ_KEY_JOBNAME

ID 0x003B

Type char[11]

Description

The name of the job that created the spooled file.

Job Number

Key CWBOBJ_KEY_JOBNUMBER

ID 0x003C

Type char[7]

Description

The number of the job that created the spooled file.

Job Separators

Key CWBOBJ_KEY_JOBSEPRATR

ID 0x003D

Type long

Description

The number of job separators to be placed at the beginning of the output for each job having spooled files on this output queue. Valid values are -2, 0-9. The value *MSG is encoded as -2. Job separators are specified when the output queue is created.

Job User

Key CWBOBJ_KEY_USER

ID 0x003E

Type char[11]

Description

The name of the user that created the spooled file.

Last Page Printed

Key CWBOBJ_KEY_LASTPAGE

ID 0x003F

Type long

Description

The number of the last printed page is the file if printing ended before the job completed processing.

Length of Page

Key CWBOBJ_KEY_PAGELEN

ID 0x004E

Type float

Description

The length of a page. Units of measurement are specified in the measurement method attribute.

Library Name

Key CWBOBJ_KEY_LIBRARY

ID 0x000F

Type char[11]

Description

The name of the library.

Lines Per Inch

Key CWBOBJ_KEY_LPI

ID 0x0040

Type float

Description

The number of lines per vertical inch in the spooled file.

Manufacturer Type and Model

Key CWBOBJ_KEY_MFGTYPE

ID 0x0041

Type char[21]

Description

Specifies the manufacturer, type, and model when transforming print data from SCS to ASCII.

Maximum Spooled Output Records

Key CWBOBJ_KEY_MAXRECORDS

ID 0x0042

Type long

Description

The maximum number of records allowed in this file at the time this file was opened. The value *NOMAX is encoded as 0.

Measurement Method

Key CWBOBJ_KEY_MEASMETHOD

ID 0x004F

Type char[11]

Description

The measurement method that is used for the length of page and width of page attributes. Valid values are *ROWCOL, *UOM.

Message Help

Key CWBOBJ_KEY_MSGHELP

ID 0x0081

Type char(*)

Description

The message help, which is sometimes known as second-level text, can be returned by a "retrieve message" request. The system limits the length to 3000 characters (English version must be 30 % less to allow for translation).

Message ID

Key CWBOBJ_KEY_MESSAGEID

ID 0x0093

Type char[8]

Description

The message ID.

Message Queue Library Name

Key CWBOBJ_KEY_MSGQUELIB

ID 0x0044

Type char[11]

Description

The name of the library that contains the message queue.

Message Queue

Key CWBOBJ_KEY_MSGQUE

ID 0x005E

Type char[11]

Description

The name of the message queue that the writer uses for operational messages.

Message Reply

Key CWBOBJ_KEY_MSGREPLY

ID 0x0082

Type char[133]

Description

The message reply. Text string to be provided by the client which answers a message of type "inquiry". In the case of message retrieved, the attribute value is returned by the server and contains the default reply which the client can use. The system limits the length to 132 characters. Should be null-terminated due to variable length.

Message Text

Key CWBOBJ_KEY_MSGTEXT

ID 0x0080

Type char[133]

Description

The message text, that is sometimes known as first-level text, can be returned by a "retrieve message" request. The system limits the length to 132 characters.

Message Type

Key CWBOBJ_KEY_MSGTYPE

ID 0x008E

Type char[3]

Description

The message type, a 2-digit, EBCDIC encoding. Two types of messages indicate whether one can "answer" a "retrieved" message: '04' Informational messages convey information without asking for a reply (may require a corrective action instead), '05' Inquiry messages convey information and ask for a reply.

Message Severity

Key CWBOBJ_KEY_MSGSEV

ID 0x009F

Type long

Description

Message severity. Values range from 00 to 99. The higher the value, the more severe or important the condition.

Number of Bytes to Read/Write

Key CWBOBJ_KEY_NUMBYTES

ID 0x007D

Type long

Description

The number of bytes to read for a read operation, or the number of bytes to write for a write operation. The object action determines how to interpret this attribute.

Number of Files

Key CWBOBJ_KEY_NUMFILES

ID 0x0045

Type long

Description

The number of spooled files that exist on the output queue.

Number of Writers Started to Queue

Key CWBOBJ_KEY_NUMWRITERS

ID 0x0091

Type long

Description

The number of writer jobs started to the output queue.

Object Extended Attribute

Key CWBOBJ_KEY_OBJEXTATTR

ID 0x000B1

Type char[11]

Description

An "extended" attribute used by some objects like font resources. This value shows up via WRKOBJ and DSPOBJD commands on the iSeries server. The title on an iSeries server screen may just indicate "Attribute". In the case of object types of font resources, for example, common values are CDEPAG, CDEFNT, and FNTCHRSET.

Open time commands

Key CWBOBJ_KEY_OPENCMDSDS

ID 0x00A0

Type char[11]

Description

Specifies whether the user wants SCS open time commands to be inserted into datastream prior to spool file data. Valid values are *YES, *NO.

Operator Controlled

Key CWBOBJ_KEY_OPCNTRL

ID 0x0046

Type char[11]

Description

Whether users with job control authority are allowed to manage or control the spooled files on this queue. Valid values are *YES, *NO.

Order of Files On Queue

Key CWBOBJ_KEY_ORDER

ID 0x0047

Type char[11]

Description

The order of spooled files on this output queue. Valid values are *FIFO, *JOBNBR.

Output Priority

Key CWBOBJ_KEY_OUTPTY

ID 0x0048

Type char[11]

Description

The priority of the spooled file. The priority ranges from 1 (highest) to 9 (lowest). Valid values are 0-9, where 0 represents *JOB.

Output Queue Library Name

Key CWBOBJ_KEY_OUTQUELIB

ID 0x0049

Type char[11]

Description

The name of the library that contains the output queue.

Output Queue Name

Key CWBOBJ_KEY_OUTQUE

ID 0x004A

Type char[11]

Description

The name of the output queue.

Output Queue Status

Key CWBOBJ_KEY_OUTQUESTS

ID 0x004B

Type char[11]

Description

The status of the output queue. Valid values are RELEASED, HELD.

Overflow Line Number

Key CWBOBJ_KEY_OVERFLOW

ID 0x004C

Type long

Description

The last line to be printed before the data that is being printed overflows to the next page.

Pages Per Side

Key CWBOBJ_KEY_MULTIUP

ID 0x0052

Type long

Description

The number of logical pages that print on each side of each physical page when the file is printed. Valid values are 1, 2, 4.

Pel Density

Key CWBOBJ_KEY_PELDENSITY

ID 0x00B2

Type char[2]

Description

For font resources only, this value is an encoding of the number of pels ("1" represents a pel size of 240, "2" represents a pel size of 320). Additional values may become meaningful as the iSeries system defines them.

Point Size

Key CWBOBJ_KEY_POINTSIZE

ID 0x0053

Type float

Description

The point size in which this spooled file's text is printed. The special value *NONE will be encoded as 0.

Print Fidelity

Key CWBOBJ_KEY_FIDELITY

ID 0x0054

Type char[11]

Description

The kind of error handling that is performed when printing. Valid values are *ABSOLUTE, *CONTENT.

Print on Both Sides

Key CWBOBJ_KEY_DUPLEX

ID 0x0055

Type char[11]

Description

How the information prints. Valid values are *FORMDF, *NO, *YES, *TUMBLE.

Print Quality

Key CWBOBJ_KEY_PRTQUALITY

ID 0x0056

Type char[11]

Description

The print quality that is used when printing this spooled file. Valid values are *STD, *DRAFT, *NLQ, *FASTDRAFT.

Print Sequence

Key CWBOBJ_KEY_PRTSEQUENCE

ID 0x0057

Type char[11]

Description

Print sequence. Valid values are *NEXT.

Print Text

Key CWBOBJ_KEY_PRTTEXT

ID 0x0058

Type char[31]

Description

The text that is printed at the bottom of each page of printed output and on separator pages. Valid special values include *BLANK and *JOB.

Printer

Key CWBOBJ_KEY_PRINTER

ID 0x0059

Type char[11]

Description

The name of the printer device.

Printer Device Type

Key CWBOBJ_KEY_PRTDEVTYPE

ID 0x005A

Type char[11]

Description

The printer data stream type. Valid values are *SCS, *IPDS(*), *USERASCII, *AFPDS.

Printer File Library Name

Key CWBOBJ_KEY_PRTRFILELIB

ID 0x005B

Type char[11]

Description

The name of the library that contains the printer file.

Printer File Name

Key CWBOBJ_KEY_PRTRFILE

ID 0x005C

Type char[11]

Description

The name of the printer file.

Printer Queue

Key CWBOBJ_KEY_RMTPTQ

ID 0x005D

Type char[129]

Description

The name of the destination printer queue when sending spooled files via SNDTCPSPLF (LPR).

Record Length

Key CWBOBJ_KEY_RECLENGTH

ID 0x005F

Type long

Description

Record length.

Remote System

Key CWBOBJ_KEY_RMTSYSTEM

ID 0x0060

Type char[256]

Description

Remote system name. Valid special values include *INTNETADR.

Replace Unprintable Characters

Key CWBOBJ_KEY_RPLUNPRT

ID 0x0061

Type char[11]

Description

Whether characters that cannot be printed are to be replaced with another character. Valid values are *YES or *NO.

Replacement Character

Key CWBOBJ_KEY_RPLCHAR

ID 0x0062

Type char[2]

Description

The character that replaces any unprintable characters.

Resource library name

Key CWBOBJ_KEY_RSCLIB

ID 0x00AE

Type char[11]

Description

The name of the library that contains the external AFP (Advanced Function Print) resource.

Resource name

Key CWBOBJ_KEY_RSCNAME

ID 0x00AF

Type char[11]

Description

The name of the external AFP resource.

Resource object type

Key CWBOBJ_KEY_RSCTYPE

ID 0x00B0

Type Long

Description

A numerical, bit encoding of external AFP resource object type. Values are 0x0001, 0x0002, 0x0004, 0x0008, 0x0010 corresponding to *FNTRSC, *FORMDF, *OVL, *PAGSEG, *PAGDFN, respectively.

Restart Printing

Key CWBOBJ_KEY_RESTART

ID 0x0063

Type long

Description

Restart printing. Valid values are -1, -2, -3, or the page number to restart at. The value *STRPAGE is encoded as -1, the value *ENDPAGE is encoded as -2, and the value *NEXT is encoded as -3.

Save Spooled File

Key CWBOBJ_KEY_SAVESPLF

ID 0x0064

Type char[11]

Description

Whether the spooled file is to be saved after it is written. Valid values are *YES, *NO.

Seek Offset

Key CWBOBJ_KEY_SEEKOFF

ID 0x007E

Type long

Description

Seek offset. Allows both positive and negative values relative to the seek origin.

Seek Origin

Key CWBOBJ_KEY_SEEKORG

ID 0x007F

Type long

Description

Valid values include 1 (beginning or top), 2 (current), and 3 (end or bottom).

Send Priority

Key CWBOBJ_KEY_SENDPTY

ID 0x0065

Type char[11]

Description

Send priority. Valid values are *NORMAL, *HIGH.

Separator page

Key CWBOBJ_KEY_SEPPAGE

ID 0x00A1

Type char[11]

Description

Allows a user the option of printing a banner page. Valid values are *YES or *NO.

Source Drawer

Key CWBOBJ_KEY_SRCDRWR

ID 0x0066

Type long

Description

The drawer to be used when the automatic cut sheet feed option is selected. Valid values are -1, -2, 1-255. The value *E1 is encode as -1, and the value *FORMDF is encoded as -2.

Spool SCS

Key CWBOBJ_KEY_SPLSCS

ID 0x00AD

Type Long

Description

Determines how SCS data is used during create spool file. Valid values are -1, 0, 1, or the page number. The value *ENDPAGE is encoded as -1. For the value 0, printing starts on page 1. For the value 1, the entire file prints.

Spool the Data

Key CWBOBJ_KEY_SPOOL

ID 0x0067

Type char[11]

Description

Whether the output data for the printer device is spooled. Valid values are *YES, *NO.

Spooled File Name

Key CWBOBJ_KEY_SPOOLFILE

ID 0x0068

Type char[11]

Description

The name of the spooled file.

Spooled File Number

Key CWBOBJ_KEY_SPLFNUM

ID 0x0069

Type long

Description

The spooled file number.

Spooled File Status

Key CWBOBJ_KEY_SPLFSTATUS

ID 0x006A

Type char[11]

Description

The status of the spooled file. Valid values are *CLOSED, *HELD, *MESSAGE, *OPEN, *PENDING, *PRINTER, *READY, *SAVED, *WRITING.

Spooled Output Schedule

Key CWBOBJ_KEY_SCHEDULE

ID 0x006B

Type char[11]

Description

Specifies, for spooled files only, when the spooled file is available to the writer. Valid values are *IMMED, *FILEEND, *JOBEND.

Starting Page

Key CWBOBJ_KEY_STARTPAGE

ID 0x006C

Type long

Description

The page number at which to start printing the spooled file. Valid values are -1, 0, 1, or the page number. The value *ENDPAGE is encoded as -1. For the value 0, printing starts on page 1. For the value 1, the entire file prints.

Text Description

Key CWBOBJ_KEY_DESCRIPTION

ID 0x006D

Type [51]

Description

Text to describe an instance of an iSeries object.

Time File Opened

Key CWBOBJ_KEY_TIMEOPEN

ID 0x006E

Type char[7]

Description

The time this spooled file was opened. The time is encoded in a character 0x0005 with the following format, HH MM SS.

Total Pages

Key CWBOBJ_KEY_PAGES

ID 0x006F

Type long

Description

The number of pages that are contained in a spooled file.

Transform SCS to ASCII

Key CWBOBJ_KEY_SCS2ASCII

ID 0x0071

Type char[11]

Description

Whether the print data is to be transformed from SCS to ASCII. Valid values are *YES, *NO.

Unit of Measure

Key CWBOBJ_KEY_UNITOFMEAS

ID 0x0072

Type char[11]

Description

The unit of measure to use for specifying distances. Valid values are *CM, *INCH.

User Comment

Key CWBOBJ_KEY_USERCMT

ID 0x0073

Type char[101]

Description

The 100 characters of user-specified comment that describe the spooled file.

User Data

Key CWBOBJ_KEY_USERDATA

ID 0x0074

Type char[11]

Description

The 10 characters of user-specified data that describe the spooled file. Valid special values include *SOURCE.

User defined data

Key CWBOBJ_KEY_USRDFNDA

ID 0x00A2

Type char[]

Description

User defined data to be utilized by user applications or user specified programs that process spool files. All characters are acceptable. Max size is 255.

User defined object library

Key CWBOBJ_KEY_USRDFNOBJLIB

ID 0x00A4

Type char[11]

Description

User defined object library to search by user applications that process spool files.

User defined object name

Key CWBOBJ_KEY_USRDFNOBJ

ID 0x00A5

Type char[11]

Description

User defined object name to be utilized by user applications that process spool files.

User defined object type

Key CWBOBJ_KEY_USRDFNOBJTYP

ID 0x00A6

Type char[11]

Description

User defined object type pertaining to the user defined object.

User defined option(s)

Key CWBOBJ_KEY_USEDFNOPTS

ID 0x00A3

Type char[*]

Description

User defined options to be utilized by user applications that process spool files. Up to 4 options may be specified, each value is length char(10). All characters are acceptable.

User driver program

Key CWBOBJ_KEY_USRDRVPGMDTA

ID 0x00A9

Type char[11]

Description

User data to be used with the user driver program. All characters are acceptable. Maximum size is 5000 characters.

User driver program library

Key CWBOBJ_KEY_USRDRVPGMLIB

ID 0x00AA

Type char[11]

Description

User defined library to search for driver program that processes spool files.

User driver program name

Key CWBOBJ_KEY_USRDRVPGM

ID 0x00AB

Type char[11]

Description

User defined program name that processes spool files.

User ID

Key CWBOBJ_KEY_TOUSERID

ID 0x0075

Type char[9]

Description

User ID to which the spooled file is sent.

User ID Address

Key CWBOBJ_KEY_TOADDRESS

ID 0x0076

Type char[9]

Description

Address of user to whom the spooled file is sent.

User transform program library

Key CWBOBJ_KEY_USRTFMPGMLIB

ID 0x00A7

Type char[11]

Description

User defined library search for transform program.

User transform program name

Key CWBOBJ_KEY_USETFMPGM

ID 0x00A8

Type char[11]

Description

User defined transform program name that transforms spool file data before it is processed by the driver program.

VM/MVS Class

Key CWBOBJ_KEY_VMMVSCCLASS

ID 0x0077

Type char[2]

Description

VM/MVS class. Valid values are A-Z and 0-9.

When to Automatically End Writer

Key CWBOBJ_KEY_WTRAUTOEND

ID 0x0078

Type char[11]

Description

When to end the writer if it is to be ended automatically. Valid values are *NORDYF, *FILEEND. Attribute Automatically end writer must be set to *YES.

When to End Writer

Key CWBOBJ_KEY_WTREND

ID 0x0090

Type char[11]

Description

When to end the writer. Valid value are *CNTRLD, *IMMED, and *PAGEEND. This is different from when to automatically end the writer.

When to Hold File

Key CWBOBJ_KEY_HOLDTYPE

ID 0x009E

Type char[11]

Description

When to hold the spooled file. Valid values are *IMMED, and *PAGEEND.

Width of Page

Key CWBOBJ_KEY_PAGEWIDTH

ID 0x0051

Type float

Description

The width of a page. Units of measurement are specified in the measurement method attribute.

Workstation Customizing Object Name

Key CWBOBJ_KEY_WSCUSTMOBJ

ID 0x0095

Type char[11]

Description

The name of the workstation customizing object.

Workstation Customizing Object Library

Key CWBOBJ_KEY_WSCUSTMOBJL

ID 0x0096

Type char[11]

Description

the name of the library that contains the workstation customizing object.

Writer Job Name

Key CWBOBJ_KEY_WRITER

ID 0x0079

Type char[11]

Description

The name of the writer job.

Writer Job Number

Key CWBOBJ_KEY_WTRJOBNUM

ID 0x007A

Type char[7]

Description

The writer job number.

Writer Job Status

Key CWBOBJ_KEY_WTRJOBSTS

ID 0x007B

Type char[11]

Description

The status of the writer job. Valid values are STR, END, JOBQ, HLD, MSGW.

Writer Job User Name

Key CWBOBJ_KEY_WTRJOBUSER

ID 0x007C

Type char[11]

Description

The name of the user that started the writer job.

Writer Starting Page

Key CWBOBJ_KEY_WTRSTRPAGE

ID 0x008F

Type long

Description

Specifies the page number of the first page to print from the first spooled file when the writer job starts. This is only valid if the spooled file name is also specified when the writer starts.

Network Print Server Object Attributes

- “NPS Attribute Default Value”
- “NPS Attribute High Limit”
- “NPS Attribute ID” on page 315
- “NPS Attribute Low Limit” on page 315
- “NPS Attribute Possible Value” on page 315
- “NPS Attribute Text Description” on page 315
- “NPS Attribute Type” on page 315
- “NPS CCSID” on page 315
- “NPS Object” on page 316
- “NPS Object Action” on page 316
- “NPS Level” on page 316

NPS Attribute Default Value:

Key CWBOBJ_KEY_ATTRDEFAULT

ID 0x0083

Type dynamic

Description

Default value for the attribute.

NPS Attribute High Limit:

Key CWBOBJ_KEY_ATTRMAX

ID 0x0084

Type dynamic

Description

High limit of the attribute value.

NPS Attribute ID:

Key CWBOBJ_KEY_ATTRID

ID 0x0085

Type long

Description

ID of the attribute.

NPS Attribute Low Limit:

Key CWBOBJ_KEY_ATTRMIN

ID 0x0086

Type dynamic

Description

Low limit of the attribute value.

NPS Attribute Possible Value:

Key CWBOBJ_KEY_ATTRPOSSIBL

ID 0x0087

Type dynamic

Description

Possible value for the attribute. More than one NPS possible value instance may be present in a code point.

NPS Attribute Text Description:

Key CWBOBJ_KEY_ATTRDESCRIPT

ID 0x0088

Type char(*)

Description

Text description that provides a name for the attribute.

NPS Attribute Type:

Key CWBOBJ_KEY_ATTRTYPE

ID 0x0089

Type long

Description

The type of the attribute. Valid values are the types that are defined by the Network Print Server.

NPS CCSID:

Key CWBOBJ_KEY_NPSCCSID

ID 0x008A

Type long

Description

CCSID that the Network Print Server expects that all strings will be encoded in.

NPS Object:

Key CWBOBJ_KEY_NPSOBJECT

ID 0x008B

Type long

Description

Object ID. Valid values are the objects that are defined by the Network Print Server.

NPS Object Action:

Key CWBOBJ_KEY_NPSACTION

ID 0x008C

Type long

Description

Action ID. Valid values are the actions that are defined by the Network Print Server.

NPS Level:

Key CWBOBJ_KEY_NPSLEVEL

ID 0x008D

Type char[7]

Description

The version, release, and modification level of the Network Print Server. This attribute is a character string encoded as VXRYMY (ie. "V3R1M0") where

X is in (0..9)
Y is in (0..9,A..Z)

iSeries Objects API for iSeries Access for Windows listing

Note: When working with handles in the following APIs, 0 never will be returned as a valid handle.

Function/type	iSeries Objects APIs for iSeries Access for Windows
List APIs	cwbOBJ_CloseList cwbOBJ_CreateListHandle cwbOBJ_DeleteListHandle cwbOBJ_GetListSize cwbOBJ_OpenList cwbOBJ_ResetListAttrsToRetrieve cwbOBJ_ResetListFilter cwbOBJ_SetListAttrsToRetrieve cwbOBJ_SetListFilter cwbOBJ_SetListFilterWithSpIF
Object APIs	cwbOBJ_CopyObjHandle cwbOBJ_DeleteObjHandle cwbOBJ_GetObjAttr cwbOBJ_GetObjAttrs cwbOBJ_GetObjHandleFromID cwbOBJ_GetObjID cwbOBJ_RefreshObj cwbOBJ_SetObjAttrs

Function/type	iSeries Objects APIs for iSeries Access for Windows
Parameter object APIs	cwbOBJ_CopyParmObjHandle cwbOBJ_CreateParmObjHandle cwbOBJ_DeleteParmObjHandle cwbOBJ_GetParameter cwbOBJ_SetParameter
Writer job APIs	cwbOBJ_EndWriter cwbOBJ_StartWriter
Output queue APIs	cwbOBJ_HoldOutputQueue cwbOBJ_PurgeOutputQueue cwbOBJ_ReleaseOutputQueue
AFP resource APIs	cwbOBJ_CloseResource cwbOBJ_CreateResourceHandle cwbOBJ_DisplayResource cwbOBJ_OpenResource cwbOBJ_OpenResourceForSpIF cwbOBJ_ReadResource cwbOBJ_SeekResource
Spooled file APIs for working with new spooled files	cwbOBJ_CloseNewSpIF cwbOBJ_CloseNewSpIFAndGetHandle cwbOBJ_CreateNewSpIF cwbOBJ_GetSpIFHandleFromNewSpIF cwbOBJ_WriteNewSpIF
Spooled file APIs for reading spooled files	cwbOBJ_CloseSpIF cwbOBJ_OpenSpIF cwbOBJ_ReadSpIF cwbOBJ_SeekSpIF
Spooled file APIs for manipulating iSeries spooled files	cwbOBJ_CallExitPgmForSpIF cwbOBJ_CreateSpIFHandle cwbOBJ_CreateSpIFHandleEx cwbOBJ_DeleteSpIF cwbOBJ_DisplaySpIF cwbOBJ_HoldSpIF cwbOBJ_IsViewerAvailable cwbOBJ_MoveSpIF cwbOBJ_ReleaseSpIF cwbOBJ_SendNetSpIF cwbOBJ_SendTCPSpIF
Spooled file APIs for handling spooled file messages	cwbOBJ_AnswerSpIFMsg cwbOBJ_GetSpIFMsgAttr
Spooled file API for analyzing data	cwbOBJ_AnalyzeSpIFData
Server program APIs	cwbOBJ_DropConnections cwbOBJ_GetNPServerAttr cwbOBJ_SetConnectionsToKeep

cwbOBJ_AnalyzeSpIFData

Purpose: Analyze data for a spooled file and give a best guess as to what the data type is.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_AnalyzeSpIFData(  
    const char *data,  
    unsigned long bufLen,  
    cwbOBJ_SpIFDataType *dataType,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

const char *data - input

pointer to data to be analyzed.

unsigned long bufLen - input

The length of the buffer pointed to by data.

cwbOBJ_SpIFDataType *dataType - output

On output this will contain the data type. If the data type can not be determined, it defaults to CWB OBJ_DT_USERASCII.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_PARAMETER

Invalid parameter specified.

Usage: This uses the same routine that is used during the creation of spooled files that don't have a data type specified or have a data type of *AUTO specified. The result defaults to *USERASCII if it can not be determined.

cwbOBJ_AnswerSpIFMsg

Purpose: Answer the message that the spooled file is waiting on.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_AnswerSpIFMsg(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        char                *msgAnswer,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to answer the message for.

const char *msgAnswer - input

Pointer to a ASCIIZ string that contains the answer for the message.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle

CWBOBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

CWBOBJ_RC_SPLFNOMESSAGE

The spooled file isn't waiting on a message.

Usage: None

cwbOBJ_CallExitPgmForSpIF

Purpose: Instructs the iSeries Access Netprint server program, QNPSEVR, to call down its exit program chain passing this spooled file's ID and some application specified data as parameters.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CallExitPgmForSpIF(  
                        cwbOBJ_ObjHandle  spIFHandle,  
                        void               *data,  
                        unsigned long      dataLen,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ObjHandle spIFHandle - input

Handle of the spooled file to be passes as a parameter to the exit programs.

void *data - input

Pointer to a block of date that will be passed to the exit programs. The format of this data is exit program specific.

unsigned long dataLen - input

length of data pointed to by pData.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWB OBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWB OBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

CWB OBJ_RC_NO_EXIT_PGM

No exit program is registered with the Network Print server.

Usage: This is a way for a client program to communicate with its server portion to do processing of spooled files. All exit programs registered with the QNPSEVR program on the iSeries server will be called, so it is up to the client program and exit program to architect the format of the data in *data such that the exit program can recognize it. See the iSeries server 'Guide to Programming for Print' for information on the interface between the QNPSEVR server program and the exit programs.

cwbOBJ_CloseNewSpIF

Purpose: Closes a newly created spooled file.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_CloseNewSpIF(  
    cwbOBJ_ObjHandle newSpIFHandle,  
    cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle newSpIFHandle - input

New spooled file handle. This is the handle passed back on the cwbOBJ_CreateNewSpIF() API.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

Usage: Once a spooled file is closed, you can no longer write to it.

cwbOBJ_CloseNewSpIFAndGetHandle

Purpose: Closes a newly created spooled file and returns a handle to it.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_CloseNewSpIFAndGetHandle(  
    cwbOBJ_ObjHandle    newSpIFHandle,  
    cwbOBJ_ObjHandle    *spIFHandle,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbOBJ_ObjHandle newSpIFHandle - input

New spooled file handle. This is the handle passed back on the cwbOBJ_CreateNewSpIF() API.

cwbOBJ_ObjHandle *spIFHandle - output

Pointer to an object handle that, upon successful, completion of this call, will hold the spooled file handle. This handle may be used with other APIs that take a spooled file handle as input.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

Usage: The handle returned in spIFHandle must be released with the cwbOBJ_DeleteObjHandle() API in order to free resources.

cwbOBJ_CloseList

Purpose: Closes an opened list.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CloseList(  
                                cwbOBJ_ListHandle  listHandle,  
                                cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

Handle of the list to be closed. This list must be opened.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated list handle.

CWBOBJ_RC_LIST_NOT_OPEN

The list isn't open.

Usage: Closing the list frees the memory used by the list to hold its items. Any object handles gotten with `cwbOBJ_GetObjHandle()` API should be released before closing the list to free resources. These handles are no longer valid.

cwbOBJ_CloseResource

Purpose: Closes an AFP Resource object that was previously opened for for reading.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CloseResource(  
                        cwbOBJ_ObjHandle  resourceHandle,  
                        cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbOBJ_ObjHandle resourceHandle - input

Handle of the resource to be closed.

cwbSV_ErrHandle errorHandler - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid resource handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandler.

CWBOBJ_RC_RSCNOTOPEN

Resource not opened.

CWBOBJ_RC_SPLFNOTOPEN

Spooled file not open.

Usage: If the handle for the resource was obtained via a call to the cwbOBJ_OpenResourceForSpIF() API, then this api will delete the handle for you (the handle was dynamically allocated for you when you opened the resource and this call deallocates it).

cwbOBJ_CloseSpIF

Purpose: Closes an iSeries spooled file that was previously opened for for reading.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CloseSpIF(  
                                cwbOBJ_ObjHandle  spIFHandle,  
                                cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle spIFHandle - input

Handle of the spooled file to be closed.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

Usage: None

cwbOBJ_CopyObjHandle

Purpose: Creates a duplicate handle to an object. Use this API to get another handle to the same iSeries object. This new handle will be valid until the cwbOBJ_DeleteObjHandle() API has been called to release it.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CopyObjHandle(  
                        cwbOBJ_ObjHandle  objectHandle,  
                        cwbOBJ_ObjHandle  *newObjectHandle,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle objectHandle - input

Handle of the object to copy.

cwbOBJ_ObjHandle *newObjectHandle - output

Upon successful completion of this call, this handle will contain the new object handle.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

Usage: If you have a handle to an object in a list and wish to maintain a handle to that object after the list has been closed this API allows you to do that. cwbOBJ_DeleteObjHandle() must be called to release resources for this handle.

cwbOBJ_CopyParmObjHandle

Purpose: Creates a duplicate parameter list object. All attribute keys and values in the parameter list object will be copied to the new parameter list object.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CopyParmObjHandle(  
                        cwbOBJ_ParmHandle  parmListHandle,  
                        cwbOBJ_ParmHandle  *newParmListHandle,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ParmHandle parmListHandle - input

Handle of the parameter list object to copy.

cwbOBJ_ParmHandle *newParmListHandle - output

Upon successful completion of this call, this handle will contain the new parameter list object handle.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

Usage: The `cwbOBJ_DeleteParmObjectHandle` API must be called to free resources allocated by this call.

cwbOBJ_CreateListHandle

Purpose: Allocates a handle for a list of objects. After a list handle has been allocated, the filter criteria may be set for the list with the `cwbOBJ_SetListFilter()` API, the list may be built with the `cwbOBJ_OpenList()` API, etc. `cwbOBJ_DeleteListHandle()` should be called to deallocate this list handle and free any resources used by it.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CreateListHandle(  
                        const char          *systemName,  
                        cwbOBJ_ListType     type,  
                        cwbOBJ_ListHandle   *listHandle,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

const char *systemName - input

Pointer to the system name contained in ASCIIZ string

cwbOBJ_ListType type - input

Type of list to allocate (eg. spooled file list, output queue list, etc).

cwbOBJ_ListHandle *listHandle - output

Pointer to a list handle that will be passed back on output. This handle is needed for other calls using the list.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage that is being used.

CWB_API_ERROR

General API failure.

Usage: Caller must call `cwbOBJ_DeleteListHandle` when done using this list handle. Typical calling sequence for retrieving a list of objects would be:

1. `cwbOBJ_CreateListHandle()`
2. `cwbOBJ_SetListFilter()` { repeated as needed }
3. `cwbOBJ_OpenList()`
4. `cwbOBJ_GetListSize()` to get the size of the list.
5. For $n=0$ to list size - 1 `cwbOBJ_GetObjHandle` for list item in position n do something with the object `cwbOBJ_DeleteObjHandle()`
6. `cwbOBJ_CloseList()` - You may go back to step 2 here.
7. `cwbOBJ_DeleteListHandle()`

cwbOBJ_CreateNewSpIF

Purpose: Creates a new spooled file on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_CreateNewSpIF(  
    const char          *systemName,  
    cwbOBJ_ParmHandle  *parmListHandle,  
    cwbOBJ_ObjHandle   *printerFileHandle,  
    cwbOBJ_ObjHandle   *outputQueueHandle,  
    cwbOBJ_ObjHandle   *newSpIFHandle,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

const char *systemName - input

Pointer to the system name contained in ASCIIZ string

cwbOBJ_ParmHandle *parmListHandle - input

Optional. A pointer to a valid parameter list object handle that contains parameters for creating the spooled file. Parameters set in this list override what is in the printer file and the *outputQueueHandle parameter.

cwbOBJ_ObjHandle *printerFileHandle - input

Optional. A pointer to a valid printer file object handle that references the printer file to be used when creating this spooled file. The printer file must exist on the same system that this spooled file is being created on.

cwbOBJ_ObjHandle *outputQueueHandle - input

Optional. A pointer to a valid output queue object handle that references the output queue that this spooled file should be created on. The output queue must exist on the same system that this spooled file is being created on. If the output queue is set in the *parmListHandle parameter (with CWB OBJ_KEY_OUTQUELIB & CWB OBJ_KEY_OUTQUE) it will override the output queue specified by this output queue handle.

cwbOBJ_ObjHandle *newSpIFHandle - output

A pointer to an object handle that will be filled in upon successful completion of this call with the newly created spooled file handle. This handle is needed to write data into and close the new spooled file.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: If the parmListHandle is NULL, or doesn't specify an attribute, the attribute is taken from the printer file used. If the output queue is specified with the *parmListHandle, this will override what is specified in the *outputQueueHandle parameter. If the output queue is not specified (not in the *parmListHandle AND outputQueueHandle is NULL), the output queue used is taken from the printer file. If the printer file is not specified (printerFileHandle is NULL), the server will use the default network print printer file, *LIBL/QNPSRPTF. The following parameter keys may be set in the pParmListHandl object:

CWBOBJ_KEY_ALIGN	- Align page
CWBOBJ_KEY_BKOVRLLIB	- Back overlay library name
CWBOBJ_KEY_BKOVRLAY	- Back overlay
CWBOBJ_KEY_BKOVL_ACR	- Back overlay offset across
CWBOBJ_KEY_BKOVL_DWN	- Back overlay offset down
CWBOBJ_KEY_CPI	- Characters Per Inch
(1)CWBOBJ_KEY_CODEPAGE	- Code page
CWBOBJ_KEY_COPIES	- Copies
CWBOBJ_KEY_DBCSDATA	- Contains DBCS Data
CWBOBJ_KEY_DBCSEXTENS	- Process DBCS Extension characters
CWBOBJ_KEY_DBCSROTATE	- DBCS character rotation
CWBOBJ_KEY_DBCSCPI	- DBCS CPI
CWBOBJ_KEY_DBCSSISO	- DBCS SO/SI spacing
CWBOBJ_KEY_DFR_WRITE	- Defer writing
CWBOBJ_KEY_ENDPAGE	- Ending page
(2)CWBOBJ_KEY_FILESEP	- File Separators
CWBOBJ_KEY_FOLDREC	- Fold records
CWBOBJ_KEY_FONTID	- Font identifier
CWBOBJ_KEY_FORMFEED	- Form feed
CWBOBJ_KEY_FORMTYPE	- Form type
CWBOBJ_KEY_FTOVRLIB	- Front overlay library name
CWBOBJ_KEY_FTOVRLAY	- Front overlay
CWBOBJ_KEY_FTOVL_ACR	- Front overlay offset across
CWBOBJ_KEY_FTOVL_DWN	- Front overlay offset down
(1)CWBOBJ_KEY_CHAR_ID	- Graphic character set ID
CWBOBJ_KEY_JUSTIFY	- Hardware Justification
CWBOBJ_KEY_HOLD	- Hold spooled file
CWBOBJ_KEY_LPI	- Lines per inch
CWBOBJ_KEY_MAXRECORDS	- Maximum spooled file records
CWBOBJ_KEY_OUTPTY	- Output priority
CWBOBJ_KEY_OUTQUELIB	- Output queue library name
CWBOBJ_KEY_OUTQUE	- Output queue
CWBOBJ_KEY_OVERFLOW	- Overflow line number
CWBOBJ_KEY_PAGELN	- Page length
CWBOBJ_KEY_MEASMETHOD	- Measurement method
CWBOBJ_KEY_PAGEWIDTH	- Page width
CWBOBJ_KEY_MULTIUP	- Logical number of pages per side
CWBOBJ_KEY_POINTSIZE	- The default font's point size
CWBOBJ_KEY_FIDELITY	- Print fidelity
CWBOBJ_KEY_DUPLEX	- Print on both sides
CWBOBJ_KEY_PRTQUALITY	- Print quality
CWBOBJ_KEY_PRTTEXT	- Print text
CWBOBJ_KEY_PRINTER	- Printer device name
CWBOBJ_KEY_PRTDEVTYPE	- Printer device type
CWBOBJ_KEY_RPLUNPRT	- Replace unprintable characters
CWBOBJ_KEY_RPLCHAR	- Replacement character
CWBOBJ_KEY_SAVESPLF	- Save spooled file after printing

CWBOBJ_KEY_SRCDRWR	- Source drawer
CWBOBJ_KEY_SPOOL	- Spool the data
CWBOBJ_KEY_SPOOLFILE	- Spool file name
CWBOBJ_KEY_SCHEDULE	- When spooled file available
CWBOBJ_KEY_STARTPAGE	- Starting page
CWBOBJ_KEY_UNITOFMEAS	- Unit of measure
CWBOBJ_KEY_USERCMT	- User comment (100 chars)
CWBOBJ_KEY_USERDATA	- User data (10 chars)
CWBOBJ_KEY_SPLSCS	- Spool SCS Data
CWBOBJ_KEY_USRDFNDA	- User defined data
(3)CWBOBJ_KEY_USRDFNOPTS	- User defined options
CWBOBJ_KEY_USRDFNOBJLIB	- User defined object library
CWBOBJ_KEY_USRDFNOBJ	- User defined object
CWBOBJ_KEY_USRDFNOBJTYP	- User defined object type

Notes:

1. Code page and graphic character set are dependent on each other. If you specify one of these, you must specify the other.
2. The special value of *FILE is not allowed when using this attribute to create a new spooled file.
3. Up to 4 user defined options may be specified.

cwbOBJ_CreateParmObjHandle

Purpose: Allocate a parameter list object handle. The parameter list object can be used to hold a list of parameters that can be passed in on other APIs.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_CreateParmObjHandle(  
    cwbOBJ_ParmHandle *parmListHandle,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ParmHandle *parmListHandle - output

Handle of the parameter object.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: The `cwbOBJ_DeleteParmObjectHandle` API must be called to free resources allocated by this call.

cwbOBJ_CreateResourceHandle

Purpose: Create a resource handle for a particular AFP resource on a specified system.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CreateResourceHandle(  
                        const char          *systemName,  
                        const char          *resourceName,  
                        const char          *resourceLibrary,  
                        cwbOBJ_AFPResourceType resourceType,  
                        cwbOBJ_ObjHandle    *objectHandle,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

const char *systemName - input

Pointer to the system name contained in an ASCII string.

const char *resourceName - input

Pointer to the name of the AFP resource.

const char *resourceLibrary - input

Pointer to the name of the iSeries library that contains the resource.

cwbOBJ_AFPResourceType resourceType - input

Specifies what type of resource this is. Must be one of the following:

- CWBOBJ_AFPRSC_FONT
- CWBOBJ_AFPRSC_FORMDEF
- CWBOBJ_AFPRSC_OVERLAY
- CWBOBJ_AFPRSC_PAGESEG
- CWBOBJ_AFPRSC_PAGEDDEF

cwbOBJ_ObjHandle *objectHandle - output

On output this will contain the resource handle.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the code page being used.

CWB_API_ERROR

General API failure.

Usage: Use this API to get a handle to a resource if you know the name library and type of resource. If you don't know either of these or want to choose from a list, use the list APIs to list AFP resources instead. This API does no checking of the AFP resource on the host. The first time this handle is used to retrieve data for the resource, a host error will be encountered if the resource file doesn't exist.

cwbOBJ_CreateSpIFHandle

Purpose: Create a spooled file handle for a particular spooled file on a specified system.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_CreateSpIFHandle(  
                        const char      *systemName,  
                        const char      *jobName,  
                        const char      *jobNumber,  
                        const char      *jobUser,  
                        const char      *spIFName,  
                        const unsigned long spIFNumber,  
                        cwbOBJ_ObjHandle *objectHandle,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

const char *systemName - input

Pointer to the system name contained in an ASCIIZ string.

const char *jobName - input

Pointer to the name of the iSeries job that created the spooled file in an ASCIIZ string.

const char *jobNumber - input

Pointer to the number of the iSeries job that created the spooled file in an ASCIIZ string.

const char *jobUser - input

Pointer to the user of the iSeries job that created the spooled file in an ASCIIZ string.

const char *spIFName - input

Pointer to the name of the spooled file in an ASCIIZ string.

const unsigned long spIFNumber - input

The number of the spooled file.

cwbOBJ_ObjHandle *objectHandle - output

On output this will contain the spooled file handle.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: This API does no checking of the spooled file on the host. The first time this handle is used to retrieve data for the spooled file, a host error will be encountered if the spooled file doesn't exist.

cwbOBJ_CreateSpIFHandleEx

Purpose: Create a spooled file handle for a particular spooled file on a specified system.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_CreateSpIFHandleEx(  
    const char          *systemName,  
    const char          *jobName,  
    const char          *jobNumber,  
    const char          *jobUser,  
    const char          *spIFName,  
    const unsigned long spIFNumber,  
    const char          *createdSystem,  
    const char          *createdDate,  
    const char          *createdTime,  
    cwbOBJ_ObjHandle   *objectHandle,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

const char *systemName - input

Pointer to the system name contained in an ASCIIZ string.

const char *jobName - input

Pointer to the name of the iSeries job that created the spooled file in an ASCIIZ string.

const char *jobNumber - input

Pointer to the number of the iSeries job that created the spooled file in an ASCIIZ string.

const char *jobUser - input

Pointer to the user of the iSeries job that created the spooled file in an ASCIIZ string.

const char *spIFName - input

Pointer to the name of the spooled file in an ASCIIZ string.

const unsigned long spIFNumber - input

The number of the spooled file.

const char *createdSystem - input

Pointer to the name of the system the spooled file was created on in an ASCIIZ string.

const char *createdDate - input

Pointer to the date the spooled file was created in an ASCIIZ string.

const char *createdTime - input

Pointer to the time the spooled file was created in an ASCIIZ string.

cwbOBJ_ObjHandle *objectHandle - output

On output this will contain the spooled file handle.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: This API does not check the spooled file on the host. The first time this handle is used to retrieve data for the spooled file, a host error will be encountered if the spooled file doesn't exist.

cwbOBJ_DeleteListHandle

Purpose: Deallocates a list handle that was previously allocated with the `cwbOBJ_CreateListHandle()` API. This will free any resources associated with the list.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteListHandle(  
                        cwbOBJ_ListHandle  listHandle,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

List handle that will be deleted.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_INVALID_HANDLE

List handle not found.

Usage: If the list associated with this handle is opened, this call will close it. If there are opened handles to objects in this list, they will no longer be valid. After this call returns successfully, the list handle is no longer valid.

cwbOBJ_DeleteObjHandle

Purpose: Releases a handle to an object.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_DeleteObjHandle(  
    cwbOBJ_ObjHandle objectHandle,  
    cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle objectHandle - input

Handle of the object to release.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

Usage: None

cwbOBJ_DeleteParmObjHandle

Purpose: Deallocate a parameter list object handle and free the resources used by it.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteParmObjHandle(  
                        cwbOBJ_ParmHandle  parmListHandle,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ParmHandle parmListHandle - input

Handle of the parameter object.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not a parameter object handle.

Usage: After this call returns successfully, the `parmListHandle` is no longer valid.

cwbOBJ_DeleteSpIF

Purpose: Delete an iSeries spooled file.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteSpIF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to be deleted.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

Usage: After this calls returns successfully, cwbOBJ_DeleteObjHandle() should be called to release the splFHandle.

cwbOBJ_DisplayResource

Purpose: Displays the specified AFP resource to the user.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_DisplayResource(  
                        cwbOBJ_ObjHandle    resourceHandle,  
                        const char          *view,  
                        const unsigned long flags,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbOBJ_ObjHandle resourceHandle - input

Handle of the AFP Resource object. It must be an overlay or a pagesegment type of resource.

const char *view - input

Optional, may be NULL. If specified, it is a pointer to an ASCII string that specifies the view to use when invoking the AFP viewer. There are two predefined views shipped with the viewer: LETTER (8.5" x 11") and SFLVIEW (132 column). Users may also add their own.

const unsigned long flags - input

Any of following bits may be set: CWBOBJ_DSPSPLF_WAIT - instructs this call to wait until the viewer process has successfully opened the resource before returning. If this bit is 0, this API will return after it starts the viewer process. If it is 1, this API will wait for the viewer to get the resource open before returning. All other bits must be set to 0.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate a temporary buffer.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_NO_VIEWER

The viewer support for ClientAccess/400 was not installed.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the code page that is being used.

CWB_API_ERROR

General API failure.

CWBOBJ_RC_INVALID_TYPE

The handle given for resourceHandle is not a handle to an overlay or pagesegment resource.

Usage: Use this API to bring up the AFP viewer on the specified AFP resource. The type of the resource must be an overlay or a pagesegment. A return code of CWB_NO_VIEWER means that the viewer component was not installed on the workstation.

cwbOBJ_DisplaySpIF

Purpose: Displays the specified spooled file to the user.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_DisplaySpIF(  
                        cwbOBJ_ObjHandle    spIFHandle,  
                        const char          *view,  
                        const unsigned long flags,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbOBJ_ObjHandle spIFHandle - input

Handle of the parameter object.

const char *view - input

Optional, may be NULL. If specified it is a pointer to an ASCII string that specifies the view to use when invoking the spooled file viewer. There are two predefined views shipped with the viewer:

1. LETTER (8.5" x 11")
2. SFLVIEW (132 column)

Users may also add their own.

const unsigned long flags - input

Any of the following bits may be set: CWBOBJ_DSPSPFLF_WAIT - instructs this call to wait until the viewer process has successfully opened the spooled file before returning. If this bit is 0, this API will return after it starts the viewer process. If it is 1, this API will wait for the viewer to get the spooled file open before returning. All other bits must be set to 0.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_NO_VIEWER

The viewer support for ClientAccess/400 was not installed.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: Use this API to bring up the AFP viewer on the specified spooled file. The AFP viewer can view AFP data, SCS data and plain ASCII text data. A return code of CWB_NO_VIEWER means that the viewer component was not installed on the workstation.

cwbOBJ_DropConnections

Purpose: Drops all unused conversations to all systems for the network print server for this process.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_DropConnections(  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

Usage: The CWBOBJ.DLL maintains a pool of available conversations to the network print server for use on the APIs. These conversations normally time out after not having been used for 10 to 20 minutes and are then dropped. This API allows the application to clean up the pool of conversations immediately without waiting for the timeout. It can also be used at the end of the process to make sure that any conversations are terminated. This API will drop all connections to all servers for this process that are not "in use." In use connections include those with open spooled files on them (for creating or reading from).

cwbOBJ_EndWriter

Purpose: Ends an iSeries writer job.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_EndWriter(  
                        cwbOBJ_ObjHandle  writerHandle,  
                        cwbOBJ_ParmHandle *parmListHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle writerHandle - input

Handle of the writer job to be stopped. This handle can be obtained by either listing writers and getting the writer handle from that list or from starting a writer and asking for the writer handle to be returned.

cwbOBJ_ParmHandle *parmListHandle - input

Optional. A pointer to a valid parameter list object handle that contains parameters for ending the writer.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

Usage: After this calls returns successfully, `cwbOBJ_DeleteObjHandle()` should be called to release the `writerHandle`. The following parameter key's may be set in the `pParmListHandl` object:

- **CWBOBJ_KEY_WTREND** - When to end the writer. May be any these special values:
 - *CNTRLD - end the writer after the current file is done printing.
 - *IMMED - end the writer immediately
 - *PAGEEND - end the writer at the end of the current page.

cwbOBJ_GetListSize

Purpose: Get the size of an opened list.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_GetListSize(  
    cwbOBJ_ListHandle listHandle,  
    unsigned long *size,  
    cwbOBJ_List_Status *listStatus,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

Handle of the list to get the size of. This list must be opened.

unsigned long *size - output

On output, this will be set to the current size of the list.

cwbOBJ_List_Status *listStatus - output

Optional, may be NULL. This will always be CWBOBJ_LISTSTS_COMPLETED for lists opened synchronously.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated list handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_LIST_NOT_OPEN

The list isn't open.

Usage: None

cwbOBJ_GetNPServerAttr

Purpose: Get an attribute of the QNPSEVR program on a specified system.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_GetNPServerAttr(  
    const char    *systemName,  
    cwbOBJ_KeyID  key,  
    void          *buffer,  
    unsigned long bufLen,  
    unsigned long *bytesNeeded,  
    cwbOBJ_DataType *keyType,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

const char *systemName - input

Pointer to the system name contained in an ASCII string.

cwbOBJ_KeyID key - input

Identifying key of the attribute to retrieve.

void *buffer - output

The buffer that will hold the attribute value. If this call returns successfully. The value of the key determines what type of data will be put into pBuffer. The type is also returned to the *keyType parameter, if provided.

unsigned long bufLen - input

The length of the buffer pointed to by pBuffer.

unsigned long *bytesNeeded - output

On output, this will be the number of bytes needed to hold result.

cwbOBJ_DataType *keyType - output

Optional, may be NULL. On output this will contain the type of data used to represent this attribute and what is stored at *buffer.

cwbSV_ErrHandle errorHandler - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_BUFFER_OVERFLOW

Buffer too small.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandler.

CWBOBJ_RC_INVALID_KEY

Key isn't valid.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: The following attributes may be retrieved from the QNPSERVER program:

- CWBOBJ_KEY_NPSCCSID - Server CCSID
- CWBOBJ_KEY_NPSLEVEL - Server code level

cwbOBJ_GetObjAttr

Purpose: Get an attribute of an object.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_GetObjAttr(  
    cwbOBJ_ObjHandle objectHandle,  
    cwbOBJ_KeyID     key,  
    void             *buffer,  
    unsigned long    bufLen,  
    unsigned long    *bytesNeeded,  
    cwbOBJ_DataType *keyType,  
    cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle objectHandle - input

Handle of the object to get the attribute for.

cwbOBJ_KeyID key - input

Identifying key of the attribute to retrieve. The CWBOBJ_KEY_XXX constants define the key ids. The type of object pointed to by objectHandle determine which keys are valid.

void *buffer - output

The buffer that will hold the attribute value, if this call returns successfully. The value of the key determines what type of data will be put into pBuffer. The type is also returned to the *keyType parameter, if provided.

unsigned long bufLen - input

The length of the buffer pointed to by pBuffer.

unsigned long *bytesNeeded - output

On output, this will be the number of bytes needed to hold result.

cwbOBJ_DataType *keyType - output

Optional, may be NULL. On output this will contain the type of data used to represent this attribute and what is stored at *buffer.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_BUFFER_OVERFLOW

Buffer too small.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_INVALID_KEY

Key isn't valid.

CWB_API_ERROR

General API failure.

Usage: The following attributes may be retrieved for these object types:

- CWBOBJ_LIST_SPLF:

CWBOBJ_KEY_AFP	- AFP resources used
CWBOBJ_KEY_ALIGN	- Align page
CWBOBJ_KEY_BKMGD_ACR	- Back margin across
CWBOBJ_KEY_BKMGD_DWN	- Back margin down
CWBOBJ_KEY_BKOVRLIB	- Back overlay library name
CWBOBJ_KEY_BKOVRLAY	- Back overlay name
CWBOBJ_KEY_BKOVLA_ACR	- Back overlay offset across
CWBOBJ_KEY_BKOVLA_DWN	- Back overlay offset down
CWBOBJ_KEY_CPI	- Characters per inch
CWBOBJ_KEY_CODEDFNTLIB	- Coded font library name
CWBOBJ_KEY_CODEDFNT	- Coded font
CWBOBJ_KEY_COPIES	- Copies (total)
CWBOBJ_KEY_COPIESLEFT	- Copies left to produce
CWBOBJ_KEY_CURPAGE	- Current page
CWBOBJ_KEY_DATE	- Date file was opened
CWBOBJ_KEY_PAGRTT	- Degree of page rotation
CWBOBJ_KEY_ENDPAGE	- Ending page
CWBOBJ_KEY_FILESEP	- File separators
CWBOBJ_KEY_FOLDREC	- Wrap text to next line
CWBOBJ_KEY_FONTID	- Font identifier to use (default)
CWBOBJ_KEY_FORMFEED	- Form feed
CWBOBJ_KEY_FORMTYPE	- Form type
CWBOBJ_KEY_FTMGD_ACR	- Front margin across
CWBOBJ_KEY_FTMGD_DWN	- Front margin down
CWBOBJ_KEY_FTOVRLIB	- Front overlay library name
CWBOBJ_KEY_FTOVRLAY	- Front overlay
CWBOBJ_KEY_FTOVLA_ACR	- Front overlay offset across
CWBOBJ_KEY_FTOVLA_DWN	- Front overlay offset down
CWBOBJ_KEY_CHAR_ID	- Graphic character set
CWBOBJ_KEY_JUSTIFY	- Hardware justification
CWBOBJ_KEY_HOLD	- Hold the spool file
CWBOBJ_KEY_JOBNAME	- Name of the job that created file
CWBOBJ_KEY_JOBNUMBER	- Number of the job that created file
CWBOBJ_KEY_USER	- Name of the user that created file
CWBOBJ_KEY_LASTPAGE	- Last page that printed
CWBOBJ_KEY_LPI	- Lines per inch
CWBOBJ_KEY_MAXRECORDS	- Maximum number of records allowed
CWBOBJ_KEY_OUTPTY	- Output priority
CWBOBJ_KEY_OUTQUELIB	- Output queue library name
CWBOBJ_KEY_OUTQUE	- Output queue
CWBOBJ_KEY_OVERFLOW	- Overflow line number
CWBOBJ_KEY_PAGELN	- Page length
CWBOBJ_KEY_MEASMETHOD	- Measurement method
CWBOBJ_KEY_PAGEWIDTH	- Page width
CWBOBJ_KEY_MULTIP	- Logical pages per physical side
CWBOBJ_KEY_POINTSIZE	- The default font's point size
CWBOBJ_KEY_FIDELITY	- The error handling when printing
CWBOBJ_KEY_DUPLEX	- Print on both sides of paper
CWBOBJ_KEY_PRTQUALITY	- Print quality
CWBOBJ_KEY_PRTTEXT	- Text printed at bottom of each page
CWBOBJ_KEY_PRTDEVTYPE	- Printer dev type (data stream type)
CWBOBJ_KEY_PRTRFILELIB	- Printer file library
CWBOBJ_KEY_PRTRFILE	- Printer file
CWBOBJ_KEY_RECLENGTH	- Record length
CWBOBJ_KEY_RPLUNPRT	- Replace unprintable characters

CWBOBJ_KEY_RPLCHAR - Character to replace unprintables
 CWBOBJ_KEY_RESTART - Where to restart printing at
 CWBOBJ_KEY_SAVESPLF - Save file after printing
 CWBOBJ_KEY_SRCDRWR - Source drawer
 CWBOBJ_KEY_SPOOLFILE - Spool file name
 CWBOBJ_KEY_SPLFNUM - Spool file number
 CWBOBJ_KEY_SPLFSTATUS - Spool file status
 CWBOBJ_KEY_STARTPAGE - Starting page to print
 CWBOBJ_KEY_TIME - Time spooled file was opened at
 CWBOBJ_KEY_PAGES - Number of pages in spool file
 CWBOBJ_KEY_UNITOFMEAS - Unit of measure
 CWBOBJ_KEY_USERCMT - User comment
 CWBOBJ_KEY_USERDATA - User data
 CWBOBJ_KEY_USRDFNDATA - User defined data
 CWBOBJ_KEY_USRDFNOPTS - User defined options
 CWBOBJ_KEY_USRDFNOBJ - User defined object
 CWBOBJ_KEY_USRDFNOBJLIB- User defined object library
 CWBOBJ_KEY_USRDFNOBJTYP- User defined object type

• **CWBOBJ_LIST_OUTQ:**

CWBOBJ_KEY_AUTHCHCK - authority to check
 CWBOBJ_KEY_DATAQUELIB - data queue library
 CWBOBJ_KEY_DATAQUE - data queue
 CWBOBJ_KEY_DESCRIPTION - text description
 CWBOBJ_KEY_DISPLAYANY - users can display any file on queue
 CWBOBJ_KEY_JOBSEPRATR - number of job separators
 CWBOBJ_KEY_NUMFILES - total spooled files on output queue
 CWBOBJ_KEY_NUMWRITERS - number of writers started to queue
 CWBOBJ_KEY_OPCNTRL - operator controlled
 CWBOBJ_KEY_ORDER - order on queue (sequence)
 CWBOBJ_KEY_OUTQUELIB - output queue library name
 CWBOBJ_KEY_OUTQUE - output queue
 CWBOBJ_KEY_OUTQUESTS - output queue status
 CWBOBJ_KEY_PRINTER - printer
 CWBOBJ_KEY_SEPPAGE - print banner page
 CWBOBJ_KEY_USRDFNDATA - user defined data
 CWBOBJ_KEY_USRDFNOBJ - user defined object
 CWBOBJ_KEY_USRDFNOBJLIB- user defined object library
 CWBOBJ_KEY_USRDFNOBJTYP- user defined object type
 CWBOBJ_KEY_USRDFNOPTS - user defined options
 CWBOBJ_KEY_USRDRVPGM - user driver program
 CWBOBJ_KEY_USRDRVPGMLIB- user driver program library
 CWBOBJ_KEY_USRDRVPGMDTA- user driver program data
 CWBOBJ_KEY_USRTFMPGM - user data transform program
 CWBOBJ_KEY_USRTFMPGMLIB- user data transform program library
 CWBOBJ_KEY_WRITER - writer job name
 CWBOBJ_KEY_WTRJOBNUM - writer job number
 CWBOBJ_KEY_WTRJOBSTS - writer job status
 CWBOBJ_KEY_WTRJOBUSER - writer job user

• **CWBOBJ_LIST_PRTD:**

CWBOBJ_KEY_AFP - AFP resources used
 CWBOBJ_KEY_CODEPAGE - code page
 CWBOBJ_KEY_DEVCLASS - device class
 CWBOBJ_KEY_DEVMODEL - device model
 CWBOBJ_KEY_DEVTYPE - device type
 CWBOBJ_KEY_DRWRSEP - drawer to use for separators
 CWBOBJ_KEY_FONTID - font identifier
 CWBOBJ_KEY_FORMFEED - form feed
 CWBOBJ_KEY_CHAR_ID - graphic character set
 CWBOBJ_KEY_MFGTYPE - manufacturer's type & model
 CWBOBJ_KEY_MSGQUELIB - message queue library
 CWBOBJ_KEY_MSGQUE - message queue
 CWBOBJ_KEY_POINTSIZE - default font's point size
 CWBOBJ_KEY_PRINTER - printer
 CWBOBJ_KEY_PRTQUALITY - print quality
 CWBOBJ_KEY_DESCRIPTION - text description

CWBOBJ_KEY_SCS2ASCII - transform SCS to ASCII
 CWBOBJ_KEY_USRDFNDATA - user defined data
 CWBOBJ_KEY_USRDFNOPTS - user defined options
 CWBOBJ_KEY_USRDFNOBJLIB- user defined object library
 CWBOBJ_KEY_USRDFNOBJ - user defined object
 CWBOBJ_KEY_USRDFNOBJTYP- user defined object type
 CWBOBJ_KEY_USRTFMPGMLIB- user data transform
 program library
 CWBOBJ_KEY_USRTFMPGM - user data transform program
 CWBOBJ_KEY_USRDRVPGMDTA- user driver program data
 CWBOBJ_KEY_USRDRVPGMLIB- user driver program library
 CWBOBJ_KEY_USRDRVPGM - user driver program

• CWBOBJ_LIST_PRTF:

CWBOBJ_KEY_ALIGN - align page
 CWBOBJ_KEY_BKMGD_ACR - back margin across
 CWBOBJ_KEY_BKMGD_DWN - back margin down
 CWBOBJ_KEY_BKOVRLIB - back side overlay library
 CWBOBJ_KEY_BKOVRLAY - back side overlay name
 CWBOBJ_KEY_BKOVL_DWN - back overlay offset down
 CWBOBJ_KEY_BKOVL_ACR - back overlay offset across
 CWBOBJ_KEY_CPI - characters per inch
 CWBOBJ_KEY_CODEDFNTLIB - coded font library name
 CWBOBJ_KEY_CODEPAGE - code page
 CWBOBJ_KEY_CODEDFNT - coded font
 CWBOBJ_KEY_COPIES - copies (total)
 CWBOBJ_KEY_DBCSDATA - contains DBCS character set data
 CWBOBJ_KEY_DBCSEXTENS - process DBCS extension
 characters
 CWBOBJ_KEY_DBCSRotate - rotate DBCS characters
 CWBOBJ_KEY_DBCSCPI - DBCS CPI
 CWBOBJ_KEY_DBCSSISO - DBCS SI/SO positioning
 CWBOBJ_KEY_DFR_WRITE - defer write
 CWBOBJ_KEY_PAGRRT - degree of page rotation
 CWBOBJ_KEY_ENDPAGE - ending page number to print
 CWBOBJ_KEY_FILESEP - number of file separators
 CWBOBJ_KEY_FOLDREC - wrap text to next line
 CWBOBJ_KEY_FONTID - Font identifier to use (default)
 CWBOBJ_KEY_FORMFEED - type of paperfeed to be used
 CWBOBJ_KEY_FORMTYPE - name of the form to be used
 CWBOBJ_KEY_FTMGN_ACR - front margin across
 CWBOBJ_KEY_FTMGN_DWN - front margin down
 CWBOBJ_KEY_FTOVRLIB - front side overlay library
 CWBOBJ_KEY_FTOVRLAY - front side overlay name
 CWBOBJ_KEY_FTOVL_ACR - front overlay offset across
 CWBOBJ_KEY_FTOVL_DWN - front overlay offset down
 CWBOBJ_KEY_CHAR_ID - graphic character set for this file
 CWBOBJ_KEY_JUSTIFY - hardware justification
 CWBOBJ_KEY_HOLD - hold the spool file
 CWBOBJ_KEY_LPI - lines per inch
 CWBOBJ_KEY_MAXRCDS - maximum number of records allowed
 CWBOBJ_KEY_OUTPTY - output priority
 CWBOBJ_KEY_OUTQUELIB - output queue library
 CWBOBJ_KEY_OUTQUE - output queue
 CWBOBJ_KEY_OVERFLOW - overflow line number
 CWBOBJ_KEY_LINES_PAGE - page length in lines per page
 CWBOBJ_KEY_PAGELN - page length in Units of Measurement
 CWBOBJ_KEY_MEASMETHOD - measurement method
 (*ROWCOL or *UOM)
 CWBOBJ_KEY_CHAR_LINE - page width in characters per line
 CWBOBJ_KEY_PAGewidth - width of page in Units of Measure

CWBOBJ_KEY_MULTIUP - logical pages per physical side
 CWBOBJ_KEY_POINTSIZE - the default font's point size
 CWBOBJ_KEY_FIDELITY - the error handling when printing
 CWBOBJ_KEY_DUPLEX - print on both sides of paper
 CWBOBJ_KEY_PRTQUALITY - print quality
 CWBOBJ_KEY_PRTTEXT - text printed at bottom of each page
 CWBOBJ_KEY_PRINTER - printer device name
 CWBOBJ_KEY_PRTDEVTYPE - printer dev type (data stream type)
 CWBOBJ_KEY_PRTRFILELIB - printer file library
 CWBOBJ_KEY_PRTRFILE - printer file
 CWBOBJ_KEY_RPLUNPRT - replace unprintable characters
 CWBOBJ_KEY_RPLCHAR - character to replace unprintables
 CWBOBJ_KEY_SAVE - save spooled file after printing
 CWBOBJ_KEY_SRCDRWR - source drawer
 CWBOBJ_KEY_SPOOL - spool the data
 CWBOBJ_KEY_SCHEDULE - when available to the writer
 CWBOBJ_KEY_STARTPAGE - starting page to print
 CWBOBJ_KEY_DESCRIPTION - text description
 CWBOBJ_KEY_UNITOFMEAS - unit of measure
 CWBOBJ_KEY_USERDATA - user data
 CWBOBJ_KEY_USRDFNDA - User defined data
 CWBOBJ_KEY_USRDFNOPTS - User defined options
 CWBOBJ_KEY_USRDFNOBJLIB - User defined object library
 CWBOBJ_KEY_USRDFNOBJ - User defined object
 CWBOBJ_KEY_USRDFNOBJTYP - User defined object type

- **CWBOBJ_LIST_WTR:**

CWBOBJ_KEY_WRITER - writer job name
 CWBOBJ_KEY_WTRJOBNUM - writer job number
 CWBOBJ_KEY_WTRJOBSTS - writer job status
 CWBOBJ_KEY_WTRJOBUSER - writer job user

- **CWBOBJ_LIST_LIB:**

CWBOBJ_KEY_LIBRARY - the library name
 CWBOBJ_KEY_DESCRIPTION - description of the library

- **CWBOBJ_LIST_RSC:**

CWBOBJ_KEY_RSCNAME - resource name
 CWBOBJ_KEY_RSCLIB - resource library
 CWBOBJ_KEY_RSCTYPE - resource object type
 CWBOBJ_KEY_OBJEXTATTR - object extended attribute
 CWBOBJ_KEY_DESCRIPTION - description of the resource
 CWBOBJ_KEY_DATE - date object was last modified
 CWBOBJ_KEY_TIME - time object was last modified

cwbOBJ_GetObjAttrs

Purpose: Get several attributes of an object.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_GetObjAttrs(  
    cwbOBJ_ObjHandle      objectHandle,  
    unsigned long         numAttrs,  
    cwbOBJ_GetObjAttrParms *getAttrParms,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbOBJ_ObjHandle objectHandle - input

Handle of the object to get the attribute for.

unsigned long numAttrs - input

number of attributes to retrieve

cwbOBJ_GetObjAttrParms *getAttrParms - input

an array of numAttrs elements that for each attribute to retrieve gives the attribute key (id), the buffer where to store the value for that attribute and the size of the buffer

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_BUFFER_OVERFLOW

Buffer too small.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_INVALID_KEY

Key isn't valid.

CWB_API_ERROR

General API failure.

Usage: See the Usage Notes in cwbOBJ_GetObjAttr to see which attribute are valid for the various types of objects.

cwbOBJ_GetObjHandle

Purpose: Get list object. This call gets a handle to an object in an opened list. The handle returned must be released with the the `cwbOBJ_DeleteObjHandle` when the caller is done with it to release resources. The handle returned is only valid while the list is opened.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_GetObjHandle(  
                        cwbOBJ_ListHandle listHandle,  
                        unsigned long    ulPosition,  
                        cwbOBJ_ObjHandle *objectHandle,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

Handle of the list to get the object handle from. This list must be opened.

unsigned long ulPosition - input

The position within the list of the object to get a handle for. It is 0 based. Valid values are 0 to the number of objects in the list - 1. You can use `cwbOBJ_GetListSize()` to get the size of the list.

cwbOBJ_ObjHandle *objectHandle - output

On return, this will contain the handle of the object.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated list handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

CWBOBJ_RC_LIST_NOT_OPEN

The list isn't open.

CWBOBJ_RC_INVALID_INDEX

The `ulPosition` is out of range.

Usage: None

cwbOBJ_GetObjHandleFromID

Purpose: Regenerate an object handle from its binary ID and type. `cwbOBJ_DeleteObjHandle()` must be called to free resources when you are done using the object handle.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_GetObjHandleFromID(  
                        void          *idBuffer,  
                        unsigned long  bufLen,  
                        cwbOBJ_ObjType objectType,  
                        cwbOBJ_ObjHandle *objectHandle,  
                        cwbSV_ErrHandle errorHandle);
```

Parameters:

void *idBuffer - input

The buffer that holds the id of this object.

unsigned long bufLen - input

The length of the data pointed to by `pIDBuffer`.

cwbOBJ_ObjType type - input

Type of object this ID is for. This must match the type of object the ID was taken from.

cwbOBJ_ObjHandle *objectHandle - output

If this call returns successfully, this will be the handle to the object. This handle should be released with the `cwbOBJ_DeleteObjHandle()` API when done using it.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_INVALID_TYPE

`objectType` is not correct.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

Usage: None

cwbOBJ_GetObjID

Purpose: Get the id of an object. This is the data that uniquely identifies this object on the server. The data gotten is not readable and is binary. It can be passed back on the cwbOBJ_GetObjHandleFromID() API to get a handle back to that object.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_GetObjID(  
    cwbOBJ_ObjHandle objectHandle,  
    void *idBuffer,  
    unsigned long bufLen,  
    unsigned long *bytesNeeded,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbOBJ_ObjHandle objectHandle - input

Handle of the object to get the ID from.

void *idBuffer - output

The buffer that will hold the ID of this object.

unsigned long bufLen - input

The length of the buffer pointed to by pIDBuffer.

unsigned long *bytesNeeded - output

On output, this will be the number of bytes needed to hold the ID.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_BUFFER_OVERFLOW

Buffer too small.

Usage: None

cwbOBJ_GetParameter

Purpose: Gets the value of a parameter in a parameter list object.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_GetParameter(  
    cwbOBJ_ParmHandle parmListHandle,  
    cwbOBJ_KeyID      key,  
    void              *buffer,  
    unsigned long     bufLen,  
    unsigned long     *bytesNeeded,  
    cwbOBJ_DataType   *keyType,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ParmHandle parmListHandle - input

Handle of the parameter object.

cwbOBJ_KeyID key - input

The id of the parameter to set.

void *buffer - output

The buffer that will hold the attribute value. If this call returns successfully. The value of the key determines what type of data will be put into pBuffer. The type is also returned to the *keyType parameter, if provided.

unsigned long bufLen - input

The length of the buffer pointed to by buffer.

unsigned long *bytesNeeded - output

On output, this will be the number of bytes needed to hold result.

cwbOBJ_DataType *keyType - output

Optional, may be NULL. On output this will contain the type of data used to represent this attribute and what is stored at *buffer.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_BUFFER_OVERFLOW

Buffer too small.

CWBOBJ_RC_KEY_NOT_FOUND

Key isn't specified in parameter list.

CWB_API_ERROR

General API failure.

Usage: None

cwbOBJ_GetSpIFHandleFromNewSpIF

Purpose: Uses a new spooled file handle to generate a spooled file handle. See notes below about using this API on a new spool file that was created with data type automatic.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_GetSpIFHandleFromNewSpIF(  
    cwbOBJ_ObjHandle    newSpIFHandle,  
    cwbOBJ_ObjHandle    *spIFHandle,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbOBJ_ObjHandle newSpIFHandle - input

New spooled file handle. This is the handle passed back on the cwbOBJ_CreateNewSpIF() API.

cwbOBJ_ObjHandle *spIFHandle - output

Pointer to an object handle that, upon successful completion of this call, will hold the spooled file handle. This handle may be used with other APIs that take a spooled file handle as input.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_SPLFNOTOPEN

Spooled file hasn't been created on the host yet.

Usage: The handle returned in spIFHandle must be released with the cwbOBJ_DeleteObjHandle() API in order to free resources.

If you are using automatic data typing for the spooled file (the attribute of CWBOBJ_KEY_PRTDEVTYPE was set to *AUTO or or wasn't specified on the cwbOBJ_CreateNewSpIF() API) then creation of the spooled file will be delayed until sufficient data has been written to the spooled file to determine the type of the data (*SCS, *AFPDS or *USERASCII). If the new spooled file is in this state when you call this API, the return code will be CWBOBJ_RC_SPLFNOTOPEN.

cwbOBJ_GetSpIFMsgAttr

Purpose: Retrieves an attribute of a message that's associated with a spooled file.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_GetSpIFMsgAttr(  
                        cwbOBJ_ObjHandle splFHandle,  
                        cwbOBJ_KeyID    key,  
                        void             *buffer,  
                        unsigned long    bufLen,  
                        unsigned long    *bytesNeeded,  
                        cwbOBJ_DataType  *keyType,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file.

cwbOBJ_KeyID key - input

Identifying key of the attribute to retrieve. The CWBOBJ_KEY_XXX constants define the key ids.

void *buffer - output

The buffer that will hold the attribute value, if this call returns successfully. The value of the key determines what type of data will be put into pBuffer. The type is also returned to the *keyType parameter, if provided.

unsigned long bufLen - input

The length of the buffer pointed to by pBuffer.

unsigned long *bytesNeeded - output

On output, this will be the number of bytes needed to hold result.

cwbOBJ_DataType *keyType - output

Optional, may be NULL. On output this will contain the type of data used to represent this attribute and what is stored at *buffer.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWB_BUFFER_OVERFLOW

Buffer too small.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_INVALID_KEY

Key isn't valid.

CWBOBJ_RC_SPLFNOMESSAGE

The spooled file isn't waiting on a message.

CWB_API_ERROR

General API failure.

Usage: The following keys are valid:

CWBOBJ_KEY_MSGTEXT	-	Message text
CWBOBJ_KEY_MSGHELP	-	Message help text
CWBOBJ_KEY_MSGREPLY	-	Message reply
CWBOBJ_KEY_MSGTYPE	-	Message type
CWBOBJ_KEY_MSGID	-	Message ID
CWBOBJ_KEY_MSGSEV	-	Message severity
CWBOBJ_KEY_DATE	-	Message date
CWBOBJ_KEY_TIME	-	Message time

Message formatting characters will appear in the message text and should be used as follows:

- &N** Force the text to a new line indented to column 2. If the text is longer than 1 line, the next lines should be indented to column 4 until the end of text or another format control character is found.
- &P** Force the text to a new line indented to column 6. If the text is longer than 1 line, the next lines should be indented to column 4 until the end of text or another format control character is found.
- &B** Force the text to a new line indented to column 4. If the text is longer than 1 line, the next lines should be indented to column 6 until the end of text or another format control character is found.

cwbOBJ_HoldOutputQueue

Purpose: Holds an iSeries output queue.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_HoldOutputQueue(  
                        cwbOBJ_ObjHandle  queueHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle queueHandle - input

Handle of the output queue to be held.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not a valid queue handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

Usage: None

cwbOBJ_HoldSpIF

Purpose: Holds a spooled file.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_HoldSpIF(  
                        cwbOBJ_ObjHandle  spIFHandle,  
                        cwbOBJ_ParmHandle *parmListHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle spIFHandle - input

Handle of the spooled file to be held.

cwbOBJ_ParmHandle *parmListHandle - input

Optional. A pointer to a valid parameter list object handle that contains parameters for holding the spooled file.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

Usage: The following parameter key may be set in the parmListHandle object:

- CWBOBJ_KEY_HOLDTYPE
- what type of hold to do. May be *"*IMMED"* or *"*PAGEEND"*. *"*IMMED"* is the default.

cwbOBJ_IsViewerAvailable

Purpose: Checks if the spooled file viewer is available.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_IsViewerAvailable(  
                                cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion (viewer is installed).

CWB_NO_VIEWER

Viewer not installed.

Usage: Use this function to test for the presence of the viewer on the workstation. If the viewer is installed this function will return `CWB_OK`. If the viewer is not available, the function will return `CWB_NO_VIEWER` and the `errorHandle` parameter (if provided) will contain an appropriate error message. Using this function, applications can check for viewer support without calling the `cwbOBJ_DisplaySpIF()` API.

cwbOBJ_MoveSpIF

Purpose: Moves an iSeries spooled file to another output queue or to another position on the same output queue.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_MoveSpIF(  
    cwbOBJ_ObjHandle    spIFHandle,  
    cwbOBJ_ObjHandle    *targetSpIFHandle,  
    cwbOBJ_ObjHandle    *outputQueueHandle,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbOBJ_ObjHandle spIFHandle - input

Handle of the spooled file to be moved.

cwbOBJ_ObjHandle *targetSpIFHandle - input

Optional. The handle of another spooled file on the same system, that specifies the spooled file to move this spooled file after. If this is specified, *outputQueueHandle is not used.

cwbOBJ_ObjHandle *outputQueueHandle - input

Optional. The handle of an output queue on the same system that specifies which output queue to move the spooled file to. The spooled file will be moved to the first position on this queue. This parameter is ignored if targetSpIFHandle is specified.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

Usage: If both targetSpIFHandle and outputQueueHandle are NULL, the spooled file will be moved to the first position on the current output queue.

cwbOBJ_OpenList

Purpose: Open the list. This actually builds the list. Caller must call the cwbOBJ_ClostList() API when done with the list to free resources. After the list is opened, the caller may use other APIs on the list to do things such as get the list size and get object handles to items in the list.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_OpenList(  
                        cwbOBJ_ListHandle  listHandle,  
                        cwbOBJ_List_OpenType openType,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

Handle of the list to open.

cwbOBJ_List_OpenType openHandle - input

Manner in which to open the list. Must be set to CWBOBJ_LIST_OPEN_SYNCH

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated list handle.

CWBOBJ_RC_LIST_OPEN

The list is already open.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_NOHOSTSUPPORT

Host doesn't support this type of list.

Usage: None

cwbOBJ_OpenResource

Purpose: Opens an AFP resource object for reading.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_OpenResource(  
                                cwbOBJ_ObjHandle resourceHandle,  
                                cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle resourceHandle - input

Handle of the AFP resource file to be opened for reading.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid resource handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_NOHOSTSUPPORT

Host doesn't support working with resources.

Usage: The resource should be closed with the cwbOBJ_CloseResource() API when done reading from it.

cwbOBJ_OpenResourceForSpIF

Purpose: Opens an AFP Resource object for reading for a spooled file that is already opened for reading. The API is useful if you are reading an AFP Spooled file and run into an external AFP Resource that you need to read. By using this API you can open that resource for reading without having to first list the resource.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_OpenResourceForSpIF(  
    cwbOBJ_ObjHandle    spIFHandle,  
    const char          *resourceName,  
    const char          *resourceLibrary,  
    unsigned long       resourceType,  
    const char          *reserved,  
    cwbOBJ_ObjHandle    *resourceHandle,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbOBJ_ObjHandle spIFHandle - input

Handle of the spooled file that is already opened for reading and that the resource will be opened against. The same conversation (and same instance of the the network print server program on the iSeries server) will be used for reading the resource and spooled file.

const char *resourceName - input

Pointer to the name of the AFP Resource in an ASCIIZ string.

const char *resourceLibrary - input

Optional, may be NULL. Pointer to the iSeries library of the AFP Resource in an ASCIIZ string. If no library is specified, the library list of the spooled file is used to search for the resource.

unsigned long resourceType - input

An unsigned long integer with one of the following bits on:

```
CWBOBJ_AFPRSC_FONT  
CWBOBJ_AFPRSC_FORMDEF  
CWBOBJ_AFPRSC_OVERLAY  
CWBOBJ_AFPRSC_PAGESEG  
CWBOBJ_AFPRSC_PAGEDEF
```

Specifies what type of resource to open.

const char *reserved -

Reserved, must be NULL.

cwbOBJ_ObjHandle *resourceHandle - output

Pointer to an OBJHandle that on successful return will contain the dynamically allocated resource handle that can be used to read, seek and eventually close the resource.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_FILE_NOT_FOUND

The resource wasn't found.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_INVALID_HANDLE

Handle is not valid resource handle.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_SPLFNOPEN

The spooled file is not opened.

CWBOBJ_RC_NOHOSTSUPPORT

Host doesn't support working with resources.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the code page being used.

CWB_API_ERROR

General API failure.

Usage: This call, if successful, will generate a temporary resource handle and return it in the resourceHandle parameter. This handle will be deleted automatically when the caller calls the **cwbOBJ_CloseResource()** API with it.

The resource should be closed with the **cwbOBJ_CloseResource()** API when done reading from it.

cwbOBJ_OpenSpIF

Purpose: Opens an iSeries spooled file for reading.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_OpenSpIF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to be opened for reading.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

Usage: The spooled file should be closed with the `cwbOBJ_CloseSpIF()` API when done reading from it.

cwbOBJ_PurgeOutputQueue

Purpose: Purges spooled files on an iSeries output queue.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_PurgeOutputQueue(  
                        cwbOBJ_ObjHandle  queueHandle,  
                        cwbOBJ_ParmHandle *parmListHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle queueHandle - input

Handle of the output queue to be purged.

cwbOBJ_ParmHandle * parmListHandle - input

Optional. A pointer to a valid parameter list object handle that contains parameters for purging the output queue.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

Usage: The parameters specified in `parmListHandle`, if provided, will specify which spooled files are purged. If `parmListHandle` is NULL, all spooled files for the current user are purged. The following parameter key's may be set in the `parmListHandle` object:

- **CWBOBJ_KEY_USER**
 - which user's spooled files to purge. May be a specific user ID, **"*ALL"** or **"*CURRENT"**. **"*CURRENT"** is the default.
- **CWBOBJ_KEY_FORMTYPE**
 - which spooled files to purge base on what formtype they have. May be a specific formtype, **"*ALL"** or **"*STD"**. **"*ALL"** is the default.
- **CWBOBJ_KEY_USERDATA**
 - which spooled files to purge base on what userdata they have. May be a specific value or **"*ALL"**. **"*ALL"** is the default.

cwbOBJ_ReadResource

Purpose: Reads bytes from the current read location.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_ReadResource(  
                        cwbOBJ_ObjHandle  resourceHandle,  
                        char               *buffer,  
                        unsigned long      bytesToRead,  
                        unsigned long      *bytesRead,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ObjHandle resourceHandle - input

Handle of the AFP resource object to be read from.

char *buffer - input

Pointer to buffer to hold the bytes read from the resource.

unsigned long bytesToRead - input

Maximum number of bytes to read. The number read may be less than this.

unsigned long *bytesRead - output

Number of bytes actually read.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

CWBOBJ_RC_RSCNOTOPEN

Resource file has not been opened yet.

CWBOBJ_RC_ENDOFFILE

The end of file was read.

Usage: The `cwbOBJ_OpenResource()` API must be called with this resource handle before this API is called OR the handle must be retrieved with a call to the `cwbOBJ_OpenResourceForSpIF()` API. If the end of file is reached when reading, the return code will be `CWBOBJ_RC_ENDOFFILE` and `bytesRead` will contain the actual number of bytes read.

cwbOBJ_ReadSpIF

Purpose: Reads bytes from the current read location.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_ReadSpIF(  
                        cwbOBJ_ObjHandle splFHandle,  
                        char             *bBuffer,  
                        unsigned long    bytesToRead,  
                        unsigned long    *bytesRead,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to be read from.

char *buffer - input

Pointer to buffer to hold the bytes read from the spooled file.

unsigned long bytesToRead - input

Maximum number of bytes to read. The number read may be less than this.

unsigned long *bytesRead - output

Number of bytes actually read.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

CWBOBJ_RC_SPLFNOTOPEN

Spooled file has not been opened yet.

CWBOBJ_RC_SPLFENDOFFILE

The end of file was read.

Usage: The `cwbOBJ_OpenSpIF()` API must be called with this spooled file handle before this API is called. If the end of file is reached when reading, the return code will be `CWBOBJ_SPLF_ENDOFFILE` and `bytesRead` will contain the actual number of bytes read.

cwbOBJ_RefreshObj

Purpose: Refreshes the object with the latest information from the iSeries server. This will ensure the attributes returned for the object are up to date.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_RefreshObj(  
                                cwbOBJ_ObjHandle  objectHandle,  
                                cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle objectHandle - input

Handle of the object to be refreshed.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

Usage: The following object types may be refreshed:

- CWBOBJ_LIST_SPLF (spooled files)
- CWBOBJ_LIST_PRTF (printer files)
- CWBOBJ_LIST_OUTQ (output queues)
- CWBOBJ_LIST_PRTD (printer devices)
- CWBOBJ_LIST_WTR (writers)

Example: Assume listHandle points to a spooled file list with at least one entry in it.

```
cwbOBJ_ObjHandle splFileHandle;  
u1RC = cwbOBJ_GetObjHandle(listHandle,  
0,  
&splFileHandle,  
NULL);  
if (u1RC == CWB_NO_ERROR)  
{  
    u1RC = cwbOBJ_RefreshObj(splFileHandle);  
    .....  
    get attributes for object  
    .....  
    u1RC = cwbOBJ_DeleteObjHandle(splFileHandle);  
}
```

cwbOBJ_ReleaseOutputQueue

Purpose: Releases an iSeries output queue.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_ReleaseOutputQueue(  
                        cwbOBJ_ObjHandle  queueHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle queueHandle - input

Handle of the output queue to be released.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not a valid queue handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

Usage: None

cwbOBJ_ReleaseSpIF

Purpose: Releases a spooled file.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_ReleaseSpIF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to be released.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

CWBOBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

Usage: None

cwbOBJ_ResetListAttrsToRetrieve

Purpose: Resets the list attributes to retrieve information to its default list.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_ResetListAttrsToRetrieve(  
                        cwbOBJ_ListHandle  listHandle,  
                        cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

List handle to reset.

cwbSV_ErrHandle errorHandler - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion

CWB_INVALID_HANDLE

Handle is not an allocated list handle.

Usage: Use this call to reset the list handle's list of attributes to retrieve after calling cwbOBJ_SetListAttrsToRetrieve().

cwbOBJ_ResetListFilter

Purpose: Resets the filter on a list to what it was when the list was first allocated (the default filter).

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_ResetListFilter(  
                        cwbOBJ_ListHandle listHandle,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

Handle of the list to have its filter reset.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not allocated list handle.

Usage: The list must be closed and reopened for the change to take affect.

cwbOBJ_SeekResource

Purpose: Moves the current read position on a resource that is open for reading.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_SeekResource(  
                        cwbOBJ_ObjHandle  resourceHandle,  
                        cwbOBJ_SeekOrigin  seekOrigin,  
                        signed_long       seekOffset,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle resourceHandle - input

Handle of the AFP resource file to be sought.

cwbOBJ_SeekOrigin seekOrigin - input

Where to seek from. Valid values are:

CWBOBJ_SEEK_BEGINNING - seek from the beginning of file

CWBOBJ_SEEK_CURRENT - seek from the current read position

CWBOBJ_SEEK_ENDING - seek from the end of the file

signed long seekOffset - input

Offset (negative or positive) from the seek origin in bytes to move the current read pointer to.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API.

If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

CWBOBJ_RC_RSCNOTOPEN

Resource has not been opened yet.

CWBOBJ_RC_SEEKOUTOFRANGE

Seek offset out of range.

Usage: The cwbOBJ_OpenResource() API must be called with this resource handle before this API is called OR the handle must be retrieved with a call to the cwbOBJ_OpenResourceForSpIF() API.

cwbOBJ_SeekSpIF

Purpose: Moves the current read position on a spooled file that is open for reading.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_SeekSpIF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbOBJ_SeekOrigin seekOrigin,  
                        signed_long      seekOffset,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to be closed.

cwbOBJ_SeekOrigin seekOrigin - input

Where to seek from. Valid values are:

- CWBOBJ_SEEK_BEGINNING - seek from the beginning of file
- CWBOBJ_SEEK_CURRENT - seek from the current read position
- CWBOBJ_SEEK_ENDING - seek from the end of the file

signed long seekOffset - input

Offset (negative or positive) from the seek origin in bytes to move the current read pointer to.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

CWBOBJ_RC_SPLFNOTOPEN

Spooled file has not been opened yet.

CWBOBJ_RC_SEEKOUTOFRANGE

Seek offset out of range.

Usage: The `cwbOBJ_OpenSpIF()` API must be called with this spooled file handle before this API is called.

cwbOBJ_SendNetSpIF

Purpose: Sends a spooled file to another user on the same system or to a remote system on the network.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_SendNetSpIF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbOBJ_ParmHandle  parmListHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to be sent.

cwbOBJ_ParmHandle parmListHandle - input

Required. A handle of a parameter list object that contains the parameters for sending the spooled file.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWB_INVALID_PARAMETER

invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

CWBOBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

Usage: The equivalent of a send net spooled file (SNDNETSPLF) command will be issued against the spooled file. The following parameter key's MUST be set in the `parmListHandle` object:

- **CWBOBJ_KEY_TOUSERID**
Specifies user ID to send the spooled file to.
- **CWBOBJ_KEY_TOADDRESS**
Specifies the remote system to send the spooled file to. ****NORMAL** is the default.

The following parameter key's may be set in the `parmListHandle` object:

- **CWBOBJ_KEY_DATAFORMAT**
Specifies the data format in which to transmit the spooled file. May be ****RCDDATA** or ****ALLDATA**. ****RCDDATA** is the default.
- **CWBOBJ_KEY_VMMVSCCLASS**
Specifies the VM/MVS SYSOUT class for distributions sent to a VM host system or to an MVS host system. May be "A" to "Z" or "0" to "9". "A" is the default.
- **CWBOBJ_KEY_SENDPTY**

Specifies the queueing priority used for this spooled file when it is being routed through a snad network. May be `"*NORMAL"` or `"*HIGH"`. `"*NORMAL"` is the default.

cwbOBJ_SendTCPSpIF

Purpose: Sends a spooled file to be printed on a remote system. This is the iSeries server version of the TCP/IP LPR command.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_SendTCPSpIF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbOBJ_ParmHandle  parmListHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to be sent.

cwbOBJ_ParmHandle parmListHandle - input

Required. A handle of a parameter list object that contains the parameters for sending the spooled file.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

CWBOBJ_RC_INVALID_TYPE

Handle is not a spooled file handle.

CWBOBJ_KEY_SEPPAGE

Specifies whether or not to print the separator page.

CWBOBJ_KEY_USRDTATFMLIB

Specifies the name of the user data transform library.

CWBOBJ_KEY_USRDTATFM

Specifies the name of the user data transform.

Usage: The equivalent of an iSeries server `send TCP/IP spooled file (SNDTCPSPLF)` command will be issued against the spooled file. The following parameter key's MUST be set in the `parmListHandl` object:

- **CWBOBJ_KEY_RMTSYSTEM**

Specifies the remote system to which the print request is sent. May be a remote system name or `**INTNETADR`.

- **CWBOBJ_KEY_RMTPTQ**

Specifies the name of the destination print queue.

The following parameter key's may be set in the parmListHandle object:

- CWBOBJ_KEY_DELETESPLF
Specifies whether to delete the spooled file after it has been successfully sent. May be `"*NO"` or `"*YES"`. `"*NO"` is the default.
- CWBOBJ_KEY_DESTOPTION
Specifies a destination-dependant option. These options will be sent to the remote system with the spooled file.
- CWBOBJ_KEY_DESTINATION
Specifies the type of system to which the spooled file is being sent. When sending to other iSeries systems, this value should be `"*AS/400"`. May also be `"*OTHER"`, `"*PSF/2"`. `"*OTHER"` is the default.
- CWBOBJ_KEY_INTERNETADDR
Specifies the internet address of the receiving system.
- CWBOBJ_KEY_MFGTYPE
Specifies the manufacturer, type and model when transforming print data for SCS to ASCII.
- CWBOBJ_KEY_SCS2ASCII
Specifies whether the print data is to be transformed for SCS to ASCII. May be `"*NO"` or `"*YES"`. `"*NO"` is the default.
- CWBOBJ_KEY_WSCUSTOMOBJ
Specifies the name of the workstation customizing object.
- CWBOBJ_KEY_WSCUSTOMOBL
Specifies the name of the workstation customizing object library.

cwbOBJ_SetConnectionsToKeep

Purpose: Set the number of connections that should be left active for a particular system. Normally, the cwbobj.dll will time out and drop connections after they have not been used for a while. With this API you can force it to leave open a certain number of connections for this system.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_SetConnectionsToKeep(  
    const char *systemName  
    unsigned int connections  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

const char *systemName - input

Pointer to the system name contained in ASCII string.

unsigned int connections - input

The number to of connections to keep open.

cwbSV_ErrHandle errorHandler - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_PARAMETER

Invalid parameter specified.

Usage: The default number of connections left open per system is 0. The connections are made per process, so this API only affects connections under the process it is called under. Setting the number of connections to be left open does not open any new connections.

cwbOBJ_SetListAttrsToRetrieve

Purpose: An optional function that may be applied to list handle before the list is opened. The purpose of doing this is to improve efficiency by allowing the `cwbOBJ_OpenList()` API to retrieve just the attributes of each object that the application will be using.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_SetListAttrsToRetrieve(  
                        cwbOBJ_ListHandle  listHandle,  
                        unsigned long      numKeys,  
                        const cwbOBJ_KeyID *keys,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

List handle to apply the list of attribute keys to.

unsigned long numKeys - input

The number of keys pointed to by the 'keys' parameter. May be 0, which means that no attributes are needed for objects in the list.

const cwbOBJ_KeyID *keys - input

An array of `numKeys` keys that are the IDs of the attributes to be retrieved for each object in the list when the list is opened.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated list handle.

CWB_INVALID_PARAMETER

Invalid parameter specified.

Usage: This call is used to provide a clue to the `cwbOBJ_OpenList()` API as to what attributes the application is interested in for the objects that are listed. Using this information, the `cwbOBJ_OpenList()` API can be more efficient. The attribute keys that are valid in the 'keys' list depend on type of object being listed (set on `cwbOBJ_CreateListHandle()`) Call `cwbOBJ_ResetListAttrsToRetrieve()` to reset the list to its default list of keys.

cwbOBJ_SetListFilter

Purpose: Sets filters for the list. This filter is applied the next time `cwbOBJ_OpenList()` is called.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_SetListFilter(  
                        cwbOBJ_ListHandle listHandle,  
                        cwbOBJ_KeyID     key,  
                        const char      *value,  
                        cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

List handle that this filter will be applied to.

cwbOBJ_KeyID key - input

The id of the filtering field to be set.

const void *value - input

The value this field should be set to.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_INVALID_HANDLE

List handle not found.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: The value of key will determine the type that is pointed to value. The length of value is determined by its type. The following filters may be set against these list types Spooled File Lists:

- **CWBOBJ_LIST_SPLF:**

- CWBOBJ_KEY_USER**

- Specifies which user's spooled files are to be listed. May be a specific user ID or one of these special values: *ALL - all users. *CURRENT - list spooled files for the current user only. *CURRENT is the default.

- CWBOBJ_KEY_OUTQUELIB**

- Specifies which libraries to search for output queues in. May be a specific name or one of these special values: "" - if the OUTQUEUE key word is *ALL, this combination will search all output queue on the system. *CURLIB - the current library *LIBL - the library list *LIBL is the default if the OUTQUE filter is not *ALL. "" is the default if the OUTQU filter is set to *ALL.

- CWBOBJ_KEY_OUTQUE**

- Specifies which output queues to search for spooled files on May be a specific name or the special value *ALL. *ALL is the default.

- CWBOBJ_KEY_FORMTYPE**

Specifies which spooled files are listed by the form type attribute that they have. May be a specific name or one of these special values: *ALL - spooled files with any form type are listed. *STD - spooled files with the form type of *STD are listed *ALL is the default.

CWBOBJ_KEY_USERDATA

Specifies which spooled files are listed by the user data that they have. May be a specific value or one of these special values: *ALL - spooled files with any user data value are listed. *ALL is the default.

Output Queue Lists:

- CWBOBJ_LIST_OUTQ:

CWBOBJ_KEY_OUTQUELIB

Specifies which libraries to search for output queues in. May be a specific name, a generic name or any of these special values: *ALL - all libraries *ALLUSER - all user-defined libraries, plus libraries containing user data and having names starting with Q *CURLIB - the current library *LIBL - the library list *USRLIBL - the user portion of the library list. *LIBL is the default.

CWBOBJ_KEY_OUTQUE

Specifies which output queues to list. May be a specific name, a generic name or *ALL. *ALL is the default.

Printer Device Description Lists:

- CWBOBJ_LIST_PRTD:

CWBOBJ_KEY_PRINTER

Specifies which printer device to list. May be a specific name, a generic name or *ALL. *ALL is the default.

Printer File Lists:

- CWBOBJ_LIST_PRTF:

CWBOBJ_KEY_PRTRFILELIB

Specifies which libraries to search for printer files in. May be a specific name, a generic name or any of these special values:

*ALL - all libraries

*ALLUSER - all user-defined libraries, plus libraries containing user data and having names starting with Q

*CURLIB - the current library

*LIBL - the library list

*USRLIBL - the user portion of the library list.

*ALL is the default.

CWBOBJ_KEY_PRTRFILE

Specifies which printer files to list. May be a specific name, a generic name or *ALL. *ALL is the default.

Writer Job Lists:

- CWBOBJ_LIST_WTR:

CWBOBJ_KEY_WRITER

Specifies which writer jobs to list. May be a specific name, a generic name or *ALL. *ALL is the default.

CWBOBJ_KEY_OUTQUELIB & CWBOBJ_KEY_OUTQUE

These filters are used together to get a list of writers activ to a particular output queue. If the OUTQUE key is specified the WRITER key is ignored. (all writers for the specified output queue are listed). If the OUTQUE key is specified and the OUTQUELIB isn't, the OUTQUEULIB will default to *LIBL - the system library list. The default is for neither of these to be specified.

Library Lists:

- CWBOBJ_LIST_LIB:

CWBOBJ_KEY_LIBRARY

Specifies which libraries to list. May be a specific name, a generic name or any of these special values:

- *ALL - all libraries
- *CURLIB - the current library
- *LIBL - the library list
- *USRLIBL - the user portion o the library list.
- *USRLIBL is the default.

- CWBOBJ_LIST_RSC:

Resources can be lists in a spooled file (lists all of the external AFP resources used by this spooled file) or in a library or set of libraries. To list resources for a spooled file, use the cwboBJ_SetListFilterWithSplF API along with the SetListFilter API for the RSCTYPE and RSCNAME attributes.

CWBOBJ_KEY_RSCLIB

Specifies which libraries to search for resources in. This filter is ignored if the list is filter by spooled file (i.e. SetListFilterWithSplF). May be a specific name, a generic name or any of these special values:

- *ALL - all libraries
- *ALLUSR - All user-defined libraries, plus libraries containing user data and having names starting with Q.
- *CURLIB - the current library
- *LIBL - the library list
- *USRLIBL - the user portion o the library list.
- *LIBL is the default.

CWBOBJ_KEY_RSCNAME

Specifies which resources to list by name. May be a specific name, a generic name or *ALL.

*ALL is the default.

CWBOBJ_KEY_RESCTYPE

Specifies which type of resources to list. May be any combination of the following bits logically OR'd together:

- CWBOBJ_AFPRSC_FONT
- CWBOBJ_AFPRSC_FORMDEF
- CWBOBJ_AFPRSC_OVERLAY
- CWBOBJ_AFPRSC_PAGESEG
- CWBOBJ_AFPRSC_PAGEDDEF

cwbOBJ_SetListFilterWithSpIF

Purpose: Sets filter for a list to a spooled file. For listing resources this limits the resources returned by the openList to those used by the spooled file.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_SetListFilterWithSpIF(  
                        cwbOBJ_ListHandle  listHandle,  
                        cwbOBJ_ObjHandle   splFHandle,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbOBJ_ListHandle listHandle - input

List handle that this filter will be applied to.

cwbOBJ_ObjHandle splFHandle - input

Handle of the spooled file to filter on.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWBOBJ_RC_INVALID_TYPE

Incorrect type of list.

CWB_INVALID_HANDLE

List handle not found or bad spooled file handle.

Usage: Filtering by spooled file is used when listing AFP resources so the list type must be CWBOBJ_LIST_RSC. If you filter resources based on a spooled file you cannot also filter based on a library or libraries. The resource library filter will be ignored if both are specified. Resetting a list filter will also reset the spooled file filter to nothing.

cwbOBJ_SetObjAttrs

Purpose: Change the attributes of the object on the server.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_SetObjAttrs(  
    cwbOBJ_ObjHandle  objectHandle,  
    cwbOBJ_ParmHandle parmListHandle,  
    cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbOBJ_ObjHandle objectHandle - input

Handle to the object that is to be changed.

cwbOBJ_ParmHandle parmListHandle - input

Handle to the parameter object which contains the attributes that are to be modified for the object.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle()` API. The messages may be retrieved through the `cwbSV_GetErrText()` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not an allocated object handle.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in `errorHandle`.

Usage: The following objects allow these attributes to be changed:

- CWBOBJ_LIST_SPLF (spooled files):

CWBOBJ_KEY_ALIGN	- Align page
CWBOBJ_KEY_BKOVRLLIB	- Back overlay library name
CWBOBJ_KEY_BKOVRLAY	- Back overlay
CWBOBJ_KEY_BKOVL_ACR	- Back overlay offset across
CWBOBJ_KEY_BKOVL_DWN	- Back overlay offset down
CWBOBJ_KEY_COPIES	- Copies
CWBOBJ_KEY_ENDPAGE	- Ending page
CWBOBJ_KEY_FILESEP	- File separators
CWBOBJ_KEY_FORMFEED	- Form feed
CWBOBJ_KEY_FORMTYPE	- Form type
CWBOBJ_KEY_FTOVRLLIB	- Front overlay library name
CWBOBJ_KEY_FTOVRLAY	- Front overlay
CWBOBJ_KEY_FTOVL_ACR	- Front overlay offset across
CWBOBJ_KEY_FTOVL_DWN	- Front overlay offset down
CWBOBJ_KEY_OUTPTY	- Output priority
CWBOBJ_KEY_OUTQUELIB	- Output queue library name
CWBOBJ_KEY_OUTQUE	- Output queue
CWBOBJ_KEY_MULTIUP	- Logical number of pages per side
CWBOBJ_KEY_FIDELITY	- Print fidelity
CWBOBJ_KEY_DUPLEX	- Print on both sides
CWBOBJ_KEY_PRTQUALITY	- Print quality

- CWBOBJ_KEY_PRTSEQUENCE - Print sequence
- CWBOBJ_KEY_PRINTER - Printer
- CWBOBJ_KEY_RESTART - Where to restart printing at
- CWBOBJ_KEY_SAVESPLF - Save spooled file after printing
- CWBOBJ_KEY_SCHEDULE - When spooled file available
- CWBOBJ_KEY_STARTPAGE - Starting page
- CWBOBJ_KEY_USERDATA - User data
- CWBOBJ_KEY_USRDFNDATA - User defined data
- CWBOBJ_KEY_USRDFNOPTS - User defined options
- CWBOBJ_KEY_USRDFNOBJLIB - User defined object library
- CWBOBJ_KEY_USRDFNOBJ - User defined object
- CWBOBJ_KEY_USRDFNOBJTYP - User defined object type
- CWBOBJ_LIST_PRTF (printer files):
 - CWBOBJ_KEY_ALIGN - Align page
 - CWBOBJ_KEY_BKMGD_ACR - Back margin offset across
 - CWBOBJ_KEY_BKMGD_DWN - Back margin offset down
 - CWBOBJ_KEY_BKOVRLIB - Back overlay library name
 - CWBOBJ_KEY_BKOVRLAY - Back overlay
 - CWBOBJ_KEY_BKOVL_ACR - Back overlay offset across
 - CWBOBJ_KEY_BKOVL_DWN - Back overlay offset down
 - CWBOBJ_KEY_CPI - Characters Per Inch
 - CWBOBJ_KEY_CODEPAGE - Code page
 - CWBOBJ_KEY_CODEDFNTLIB - Coded font library name
 - CWBOBJ_KEY_CODEDFNT - Coded font name
 - CWBOBJ_KEY_COPIES - Copies
 - CWBOBJ_KEY_DBCSDATA - Contains DBCS Data
 - CWBOBJ_KEY_DBCSEXTENS - Process DBCS Extension characters
 - CWBOBJ_KEY_DBCSRotate - DBCS character rotation
 - CWBOBJ_KEY_DBCSCPI - DBCS CPI
 - CWBOBJ_KEY_DBCSSISO - DBCS SO/SI spacing
 - CWBOBJ_KEY_DFR_WRITE - Defer writing
 - CWBOBJ_KEY_ENDPAGE - Ending page
 - CWBOBJ_KEY_FILESEP - File Separators(*FILE not allowed)
 - CWBOBJ_KEY_FOLDREC - Fold records
 - CWBOBJ_KEY_FONTID - Font identifier
 - CWBOBJ_KEY_FORMFEED - Form feed
 - CWBOBJ_KEY_FORMTYPE - Form type
 - CWBOBJ_KEY_FTMGN_ACR - Front margin offset across
 - CWBOBJ_KEY_FTMGN_DWN - Front margin offset down
 - CWBOBJ_KEY_FTOVRLIB - Front overlay library name
 - CWBOBJ_KEY_FTOVRLAY - Front overlay
 - CWBOBJ_KEY_FTOVL_ACR - Front overlay offset across
 - CWBOBJ_KEY_FTOVL_DWN - Front overlay offset down
 - CWBOBJ_KEY_CHAR_ID - Graphic character set ID
 - CWBOBJ_KEY_JUSTIFY - Hardware Justification
 - CWBOBJ_KEY_HOLD - Hold spooled file
 - CWBOBJ_KEY_LPI - Lines per inch
 - CWBOBJ_KEY_MAXRECORDS - Maximum spooled file records
 - CWBOBJ_KEY_OUTPTY - Output priority
 - CWBOBJ_KEY_OUTQUELIB - Output queue library name
 - CWBOBJ_KEY_OUTQUE - Output queue
 - CWBOBJ_KEY_OVERFLOW - Overflow line number
 - CWBOBJ_KEY_PAGELN - Page Length
 - CWBOBJ_KEY_MEASMETHOD - Measurement method
 - CWBOBJ_KEY_PAGewidth - Page width
 - CWBOBJ_KEY_MULTIP - Logical number of pages per side
 - CWBOBJ_KEY_POINTSIZE - The default font's point size
 - CWBOBJ_KEY_FIDELITY - Print fidelity
 - CWBOBJ_KEY_DUPLEX - Print on both sides

- CWBOBJ_KEY_PRTQUALITY - Print quality
 - CWBOBJ_KEY_PRTTEXT - Print text
 - CWBOBJ_KEY_PRINTER - Printer
 - CWBOBJ_KEY_PRTDEVTYPE - Printer Device Type
 - CWBOBJ_KEY_RPLUNPRT - Replace unprintable characters
 - CWBOBJ_KEY_RPLCHAR - Replacement character
 - CWBOBJ_KEY_SAVESPLF - Save spooled file after printing
 - CWBOBJ_KEY_SRCDRWR - Source drawer
 - CWBOBJ_KEY_SPOOL - Spool the data
 - CWBOBJ_KEY_SCHEDULE - When spooled file available
 - CWBOBJ_KEY_STARTPAGE - Starting page
 - CWBOBJ_KEY_DESCRIPTION - Text description
 - CWBOBJ_KEY_UNITOFMEAS - Unit of measure
 - CWBOBJ_KEY_USERDATA - User data
 - CWBOBJ_KEY_USRDFNDA - User defined data
 - CWBOBJ_KEY_USRDFNOPTS - User defined options
 - CWBOBJ_KEY_USRDFNOBJLIB - User defined object library
 - CWBOBJ_KEY_USRDFNOBJ - User defined object
 - CWBOBJ_KEY_USRDFNOBJTYP - User defined object type
 - CWBOBJ_LIST_OUTQ (output queues):
 - CWBOBJ_LIST_PRTD (printer devices):
 - CWBOBJ_LIST_WTR (writers):
 - CWBOBJ_LIST_LIB (libraries):
- NONE

cwbOBJ_SetParameter

Purpose: Sets the value of a parameter in a parameter list object.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_SetParameter(  
    cwbOBJ_ParmHandle parmListHandle,  
    cwbOBJ_KeyID      key,  
    const void        *value,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ParmHandle parmListHandle - input

Handle of the parameter object.

cwbOBJ_KeyID key - input

The id of the parameter to set.

void *value - input

The value to set the parameter to. The type that value points to is determined by the value of key.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not a parameter object handle.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: None

cwbOBJ_StartWriter

Purpose: Starts an iSeries writer job.

Syntax:

```
unsigned int CWB_ENTRY  cwbOBJ_StartWriter(  
                        cwbOBJ_ObjHandle  *printerHandle,  
                        cwbOBJ_ObjHandle  *outputQueueHandle,  
                        cwbOBJ_ParmHandle *parmListHandle,  
                        cwbOBJ_ObjHandle  *writerHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbOBJ_ObjHandle *printerHandle - input

Required. A pointer to a valid printer object handle that identifies which printer this writer is to be started to.

cwbOBJ_ObjHandle *outputQueueHandle - input

Optional. A pointer to a valid output queue object handle that identifies which output queue this writer is to be started from. If the parmListHandle is also specified and contains the CWBOBJ_KEY_OUTQUE parameter key, this parameter is ignored.

cwbOBJ_ParmHandle *parmListHandle - input

Optional. A pointer to a valid parameter list object handle that contains parameters for starting the writer.

cwbOBJ_ObjHandle *writerHandle - output

Optional. A pointer to a writer object handle that will be filled in upon successful return from this API. If this parameter is specified, the caller must call cwbOBJ_DeleteObjHandle() to release resources allocated for this writer handle.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid.

CWB_INVALID_PARAMETER

Invalid parameter specified.

CWBOBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

Usage: Calling this API causes the writer job to be submitted to run. The writer job may fail to start even though this API returns successfully (the job may be successfully submitted, but fail to start). This is the behavior of the STRPRTWTR command on the iSeries server. The following parameter keys may be set in the parmListHandle object:

```
CWBOBJ_KEY_ALIGN      - Align page  
CWBOBJ_KEY_ALWDRTprt  - Allow direct printing  
CWBOBJ_KEY_AUTOEND    - Automatically end writer (*YES,*NO)  
CWBOBJ_KEY_DRWRSEP    - Drawer to use for separators
```


CWBOBJ_KEY_FILESEP	- Number of file separators
CWBOBJ_KEY_FORMTYPE	- Name of the form to be used
CWBOBJ_KEY_JOBNAME	- Name of the job that created file
CWBOBJ_KEY_JOBNUMBER	- Number of the job that created file
CWBOBJ_KEY_USER	- Name of the user that created file
CWBOBJ_KEY_FORMTYPEMSG	- Form type message option
CWBOBJ_KEY_MSGQUELIB	- Message queue library
CWBOBJ_KEY_MSGQUE	- Message queue name
CWBOBJ_KEY_OUTQUELIB	- Output queue library
CWBOBJ_KEY_OUTQUE	- Output queue
CWBOBJ_KEY_SPOOLFILE	- Spool file name
CWBOBJ_KEY_SPLFNUM	- Spool file number
CWBOBJ_KEY_WTRSTRPAGE	- Page to start the writer on
CWBOBJ_KEY_WTREND	- When to end the writer
CWBOBJ_KEY_WRITER	- Writer job name
CWBOBJ_KEY_WTRINIT	- When to initialize the printer device

cwbOBJ_WriteNewSpIF

Purpose: Writes data into a newly created spooled file.

Syntax:

```
unsigned int CWB_ENTRY cwbOBJ_WriteNewSpIF(  
    cwbOBJ_ObjHandle newSpIFHandle,  
    const char      *data,  
    unsigned long   dataLen,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbOBJ_ObjHandle newSpIFHandle - input

New spooled file handle. This is the handle passed back on the cwbOBJ_CreateNewSpIF() API.

const char *data - input

Pointer to the data buffer that will be written into the spooled file.

unsigned long ulDataLen - input

Length of the data to be written.

cwbSV_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle() API. The messages may be retrieved through the cwbSV_GetErrText() API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_NO_ERROR

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_HANDLE

Handle is not valid spooled file handle.

CWB OBJ_RC_HOST_ERROR

Host error occurred. Text may be in errorHandle.

Usage: None

Example: Using iSeries Objects APIs for iSeries Access for Windows

The following example shows a typical calling sequence for retrieving a list of spooled files:

```
/******  
/* List all spooled files for the current user and      */  
/* display them to the user.                          */  
/******  
  
#ifdef UNICODE  
#define _UNICODE  
#endif  
#include <windows.h>  
  
#include <stdio.h>  
#include "CWBOBJ.H"  
main(int argc, char *argv[ ], char *envp[ ])  
{  
    cwbOBJ_ListHandle listHandle;  
    cwbOBJ_ObjHandle splFHandle;  
    unsigned int ulRC;  
    unsigned long ulListSize, ulObjPosition, ulBytesNeeded;  
    cwbOBJ_KeyID keysWanted[] = { CWBOBJ_KEY_SPOOLFILE,  
                                CWBOBJ_KEY_USER };  
    unsigned long ulNumKeysWanted = sizeof(keysWanted)/sizeof(*keysWanted);  
    char szSplFName[11];  
    char szUser[11];  
  
    ulRC = cwbOBJ_CreateListHandle(_TEXT("ANYAS400"),  
                                CWBOBJ_LIST_SPLF,  
                                &listHandle,  
                                0);  
  
    if (ulRC == CWB_OK)  
    {  
  
        /* Set up the filter for the list to be opened with */  
        /* NOTE: this is just for example, the user defaults */  
        /* to *CURRENT, so this isn't really needed.      */  
  
        cwbOBJ_SetListFilter(listHandle, CWBOBJ_KEY_USER,  
                            _TEXT("*CURRENT"), 0);  
  
        /* Optionally call to cwbOBJ_SetListAttrsToRetrieve to*/  
        /* make walking the list faster                       */  
        ulRC = cwbOBJ_SetListAttrsToRetrieve(listHandle,  
                                            ulNumKeysWanted,  
                                            keysWanted,  
                                            0);  
  
        /* open the list - this will build the list of spooled*/  
        /* files.                                             */  
        ulRC = cwbOBJ_OpenList(listHandle,  
                               CWBOBJ_LIST_OPEN_SYNCH,  
                               0);  
  
        if (ulRC == CWB_OK)  
        {  
            /* Get the number of items that are in the list */  
            ulRC = cwbOBJ_GetListSize(listHandle,  
                                     &ulListSize,  
                                     (cwbOBJ_List_Status *)0,  
                                     0);  
        }  
    }  
}
```

```

if (u1RC == CWB_OK)
{
    /* walk through the list of items, displaying */
    /* each item to the user */

    u1ObjPosition = 0;
    while (u1ObjPosition < u1ListSize)
    {
        /******
        /* Get a handle to the next spooled file in*/
        /* the list. This handle is valid while */
        /* the list is open. If you want to */
        /* maintain a handle to the spooled file */
        /* after the list is closed, you could call*/
        /* cwBOBJ_CopyObjHandle() after this call. */
        /******
        u1RC = cwBOBJ_GetObjHandle(listHandle,
                                u1ObjPosition,
                                &sp1FHandle,
                                0);

        if (u1RC == CWB_OK)
        {

            /******
            /* call cwBOBJ_GetObjAttr() to get info */
            /* about this spooled file. May also */
            /* call spooled file specific APIs */
            /* with this handle, such as */
            /* cwBOBJ_HoldSp1F(). */
            /******

            u1RC = cwBOBJ_GetObjAttr(sp1FHandle,
                                    CWBOBJ_KEY_SPOOLFILE,
                                    (void *)szSp1FName,
                                    sizeof(szSp1FName),
                                    &u1BytesNeeded,
                                    NULL,
                                    0);

            if (u1RC == CWB_OK)
            {
                u1RC = cwBOBJ_GetObjAttr(sp1FHandle,
                                        CWBOBJ_KEY_USER,
                                        (void *)szUser,
                                        sizeof(szUser),
                                        &u1BytesNeeded,
                                        NULL,
                                        0);

                if (u1RC == CWB_OK)
                {
                    printf("%3u: %11s %s\n",
                            u1ObjPosition, szSp1FName, szUser);
                } else {
                    /* ERROR on GetObjAttr! */
                }
            } else {
                /* ERROR on GetObjAttr! */
            }
            /* free this object handle */
            cwBOBJ_DeleteObjHandle(sp1FHandle, 0);

```

```

        } else {
            /* ERROR on GetObjHandle! */
        }
        u1objPosition++;
    }
} else {
    /* ERROR on GetListSize! */
}
    cwboBJ_CloseList(listHandle, 0);
} else {
    /* ERROR on OpenList! */
}

    cwboBJ_DeleteListHandle(listHandle, 0);
}

```

iSeries Access for Windows Remote Command/Distributed Program Call APIs

iSeries Access for Windows Remote Command APIs:

The iSeries Access for Windows Remote Command application programming interfaces (APIs) enable your PC application to start non-interactive commands on the iSeries system and to receive completion messages from these commands. The iSeries server command can send up to ten reply messages.

iSeries Access for Windows Distributed Program Call API:

The iSeries Access for Windows Distributed Program Call API allows your PC application to call any iSeries program or command. Input, output and in/out parameters are handled through this function. If the program runs correctly, the output and the in/out parameters will contain the data returned by the iSeries program that was called. If the program fails to run correctly on the iSeries server, the program can send up to ten reply messages.

The iSeries Access for Windows Remote Command/Distributed Program Call APIs allow the PC application programmer to access functions on the iSeries system. User program and system commands can be called without requiring an emulation session. A single iSeries program serves commands and programs, so only one iSeries job is started for both.

iSeries Access for Windows Remote Command/Distributed Program Call APIs required files:

Header file	Import library	Dynamic Link Library
cwbrc.h	cwbapi.lib	cwbrc.dll

Programmer's Toolkit:

The Programmer's Toolkit provides Remote Command and Distributed Program Call documentation, access to the cwbrc.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select either **Remote Command** or **Distributed Program Call** → **C/C++ APIs**.

iSeries Access for Windows Remote Command/Distributed Program Call APIs topics:

- "Typical use of iSeries Access for Windows Remote Command/Distributed Program Call APIs" on page 400
- **iSeries Access for Windows Remote Command/Distributed Program Call APIs listing**
- "Example: Using Remote iSeries Access for Windows Command/Distributed Program Call APIs" on page 420
- "Remote Command/Distributed Program Call APIs return codes" on page 28

Related topics:

- "iSeries system name formats for ODBC Connection APIs" on page 12

- “OEM, ANSI, and Unicode considerations” on page 12

Typical use of iSeries Access for Windows Remote Command/Distributed Program Call APIs

An application that uses the iSeries Access for Windows Remote Command/Distributed Program Call function uses objects. Each of these objects are identified to the application through a handle:

System object

This represents an iSeries system. The handle to the system object is provided to the StartSysEx function to identify the system on which the commands or APIs will be run.

Command request object

This represents the request to the iSeries system. Commands can be run and programs can be called on this object.

Note: The Command Request object previously was known as the “system object” in iSeries Access for Windows.

Program object

This represents the iSeries program. Parameters can be added, and the program can be sent to the system to run the program.

There is not a separate object for commands. The command string is sent directly to the command request.

An application that uses the Remote Command/Distributed Program Call APIs first creates a system object by calling the “cwbCO_CreateSystem” on page 61 function. This function returns a handle to the system object. This handle then is used with the “cwbRC_StartSysEx” on page 418 function to start a conversation with the iSeries system. The **cwbRC_StartSysEx** function returns a handle to the command request. Use the command request handle to call programs or to run commands. The APIs that are associated with the command request object are:

“cwbRC_StartSysEx” on page 418

“cwbRC_CallPgm” on page 404

“cwbRC_RunCmd” on page 413

“cwbRC_StopSys” on page 419

A command is a character string that is to be run on the iSeries system. Because it is a simple object (a character string) no additional object will need to be created in order to run a command. The command string simply is a parameter on the **cwbRC_RunCmd** API.

A program is a complex object that is created with the **cwbRC_CreatePgm** API, which requires the program name and the library name as parameters. The handle that is returned by this function can have 0 to 35 parameters associated with it. Parameters are added with the **cwbRC_AddParm** function. Parameters types can be input, output, or input/output. These parameters need to be in a format with which the iSeries program can work (that is, one for which no data transform or data conversion will occur). When all of the parameters have been added, the program handle is used with the **cwbRC_CallPgm** API on the command request object. The APIs that are associated with the program object are:

“cwbRC_CreatePgm” on page 405

“cwbRC_AddParm” on page 402

“cwbRC_GetParmCount” on page 411

“cwbRC_GetParm” on page 410

“cwbRC_GetPgmName” on page 412

“cwbRC_GetLibName” on page 409

“cwbRC_SetParm” on page 415

“cwbRC_SetPgmName” on page 417

“cwbRC_SetLibName” on page 414

“cwbRC_DeletePgm” on page 406

iSeries Access for Windows Remote Command/Distributed Program Call APIs listing

The following iSeries Access for Windows Remote Command/Distributed Program Call APIs are listed alphabetically, and are grouped according to function:

Function	iSeries Access for WindowsRemote Command/Distributed Program Call APIs
Access the remote command server program on the iSeries system. The request handle is used to run commands and to call programs.	cwbRC_StartSysEx cwbRC_GetClientCCSID cwbRC_GetHostCCSID cwbRC_StopSys
Run an iSeries command.	cwbRC_RunCmd
Access programs and their parameters.	cwbRC_CreatePgm cwbRC_AddParm cwbRC_CallPgm cwbRC_GetParmCount cwbRC_GetParm cwbRC_GetPgmName cwbRC_GetLibName cwbRC_SetParm cwbRC_SetPgmName cwbRC_SetLibName cwbRC_DeletePgm

cwbRC_AddParm

Purpose: Add a parameter to the program that is identified by the handle. This function should be called once for each parameter that is to be added to the program. When the program is called the parameters will be in the same order that they are added using this function.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_AddParm(  
    cwbRC_PgmHandle    program,  
    unsigned short     type,  
    unsigned long      length,  
    const unsigned char *parameter);
```

Parameters:

cwbRC_PgmHandle program - input

Handle that was returned by a previous call to the `cwbRC_CreatePgm` API. It identifies the program object.

unsigned short type - input

The type of parameter this is. Use one of the defined parameter types: `CWBRC_INPUT`, `CWBRC_OUTPUT`, `CWBRC_INOUT`. If you want to automatically convert between local CCSID and host CCSID, add the appropriate convert flag to this field with a bitwise, or use one of the defined parameter types:

```
CWBRC_TEXT_CONVERT  
CWBRC_TEXT_CONVERT_INPUT  
CWBRC_TEXT_CONVERT_OUTPUT
```

The last two types are intended for use with `CWBRC_INOUT` when conversion is only needed in one direction.

unsigned long length - input

The length of the parameter. If this is an `CWBRC_OUTPUT` parameter, the length should be the length of the buffer where the returned parameter will be written.

const unsigned char * parameter - input

Pointer to a buffer that will contain: the value if the type is `CWBRC_INPUT` or `CWBRC_INOUT`, or the place where the returned parameter is to be written if the type is `CWBRC_OUTPUT` or `CWBRC_INOUT`.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWBRC_INVALID_TYPE

Invalid type specified.

CWBRC_INVALID_PARM_LENGTH

Invalid parameter length.

CWBRC_INVALID_PARM

Invalid parameter.

Usage: Parameter data is assumed to be binary. No conversion will be performed on the parameter data unless one of the conversion flags is set. For example:


```
cwBRC_AddParm( hPgm,  
CWBRM_INOUT | CWBRM_TEXT_CONVERT_OUTPUT,  
bufferSize,  
buffer );
```

will use the buffer as is to send to the host, and will convert the output (eg to ASCII) before putting the result into the buffer.

cwbRC_CallPgm

Purpose: Calls the program identified by the handle. The return code will indicate the success or failure of the program. Additional messages can be returned by using the message handle that is returned.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_CallPgm(  
    cwbRC_SysHandle    system,  
    cwbRC_PgmHandle    program,  
    cwbSV_ErrHandle    msgHandle);
```

Parameters:

cwbRC_SysHandle system - input

Handle that was returned by a previous call to the `cwbRC_StartSysEx` function. It identifies the iSeries system.

cwbRC_PgmHandle program - input

Handle that was returned by a previous call to the `cwbRC_CreatePgm` API. It identifies the program object. object.

cwbSV_ErrHandle msgHandle - output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrTextIndexed` API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWBRC_INVALID_SYSTEM_HANDLE

Invalid system handle.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWBRC_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBRC_USER_EXIT_ERROR

Error in user exit program.

CWBRC_PROGRAM_NOT_FOUND

Program not found.

CWBRC_PROGRAM_ERROR

Error when calling program.

Usage: None

cwbRC_CreatePgm

Purpose: This function creates a program object given a program and library name. The handle that is returned can be used to add parameters to the program and then call the program.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_CreatePgm(  
    const char      *programName,  
    const char      *libraryName,  
    cwbRC_PgmHandle *program);
```

Parameters:

const char *programName - input

Pointer to an ASCII string that contains the name of the program that you want to call.

const char *libraryName - input

Pointer to an ASCII string that contains the name of the library where the program resides.

cwbRC_PgmHandle * program - output

Pointer to a cwbRC_PgmHandle where the handle of the program will be returned.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_PROGRAM_NAME

Program name is too long.

CWBRC_LIBRARY_NAME

Library name is too long.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: You should create a separate program object for each program you want to call on the iSeries server. You can use the functions described in this file to change the values of the parameters being sent to the program, but cannot change the number of parameters being sent.

cwbRC_DeletePgm

Purpose: This function deletes the program object that is identified by the handle provided.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_DeletePgm(  
    cwbRC_PgmHandle    program);
```

Parameters:

cwbRC_PgmHandle program - input

Handle that was returned by a previous call to the cwbRC_CreatePgm API. It identifies the program object.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBRC_INVALID_PROGRAM

Invalid program handle.

Usage: None.

cwbRC_GetClientCCSID

Purpose: Get the coded character set identifier (CCSID) associated with the current process. This CCSID along with the host CCSID can be used to convert EBCDIC data returned by some iSeries program to ASCII data that can be used in client applications.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_GetClientCCSID(  
    cwbRC_SysHandle    system,  
    unsigned long      *clientCCSID);
```

Parameters:

cwbRC_SysHandle system - input

Handle that was returned by a previous call to the `cwbRC_StartSysEx` function. It identifies the iSeries server system.

unsigned long * clientCCSID - output

Pointer to an unsigned long where the client CCSID will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_INVALID_SYSTEM_HANDLE

Invalid system handle.

Usage: See related APIs in the `CWBNLCNV.H` file.

cwbRC_GetHostCCSID

Purpose: Get the coded character set identifier (CCSID) associated with the iSeries server job. This CCSID along with the client CCSID can be used to convert EBCDIC data returned by some iSeries programs to ASCII data that can be used in client applications.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_GetHostCCSID(  
    cwbRC_SysHandle system,  
    unsigned long *hostCCSID);
```

Parameters:

cwbRC_SysHandle system - input

Handle that was returned by a previous call to the `cwbRC_StartSysEx` function. It identifies the iSeries system.

unsigned long * hostCCSID - output

Pointer to an unsigned long where the host CCSID will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_INVALID_SYSTEM_HANDLE

Invalid system handle.

Usage: See related APIs in the `CWBNLCONV.H` file.

cwbRC_GetLibName

Purpose: Get the name of the library that was used when creating this program object.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_GetLibName(  
    cwbRC_PgmHandle    program,  
    char                *libraryName);
```

Parameters:

cwbRC_PgmHandle program - input

Handle that was returned by a previous call to the cwbRC_CreatePgm API. It identifies the program object.

char * libraryName - output

Pointer to a ten character buffer where the name of the library will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate the temporary buffer.

CWB_API_ERROR

General API failure.

Usage: None

cwbRC_GetParm

Purpose: Retrieve the parameter identified by the index. The index will range from 0 to the total number of parameters - 1. This number can be obtained by calling the cwbRC_GetParmCount API.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_GetParm(  
    cwbRC_PgmHandle    program,  
    unsigned short     index,  
    unsigned short     *type,  
    unsigned long      *length,  
    unsigned char      **parameter);
```

Parameters:

cwbRC_PgmHandle handle - input

Handle that was returned by a previous call to the cwbRC_CreatePgm API. It identifies the program object.

unsigned short index - input

The number of the specific parameter in this program that should be retrieved. This index is zero-based.

unsigned short * type - output

Pointer to the type of parameter this is. The value will be one of the defined parameter types:

CWBRC_INPUT
CWBRC_OUTPUT
CWBRC_INOUT

unsigned long * length - input

Pointer to the length of the parameter.

unsigned char ** parameter - output

Pointer to a buffer that will contain the address of the actual parameter.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWBRC_INDEX_RANGE_ERROR

Index is out of range.

Usage: None

cwbRC_GetParmCount

Purpose: Get the number of parameters for this program object.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_GetParmCount(  
    cwbRC_PgmHandle    program,  
    unsigned short     *count);
```

Parameters:

cwbRC_PgmHandle handle - input

Handle that was returned by a previous call to the `cwbRC_CreatePgm` API. It identifies the program object.

unsigned short * count - output

Pointer to an unsigned short where the parameter count will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_INVALID_PROGRAM

Invalid program handle.

Usage: None

cwbRC_GetPgmName

Purpose: Get the name of the program that was used when creating this program.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_GetPgmName(  
    cwbRC_PgmHandle    program,  
    char                *programName);
```

Parameters:

cwbRC_PgmHandle program - input

Handle that was returned by a previous call to the cwbRC_CreatePgm API. It identifies the program object.

char * programName - output

Pointer to a ten character buffer where the name of the program will be written.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate the temporary buffer.

CWB_API_ERROR

General API failure.

Usage: None

cwbRC_RunCmd

Purpose: Issues the command on the system identified by the handle. The return code will indicate success or failure of the command. Additional messages can be returned by using the message handle that is returned.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_RunCmd(  
    cwbRC_SysHandle    system,  
    const char         *commandString,  
    cwbSV_ErrHandle    msgHandle);
```

Parameters:

cwbRC_SysHandle system - input

Handle that was returned by a previous call to the cwbRC_StartSysEx function. It identifies the iSeries system.

const char *commandString - input

Pointer to a string that contains the command to be issued on the iSeries system. This is an ASCIIZ string.

cwbSV_ErrHandle msgHandle - output

Any messages returned from the iSeries server will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrTextIndexed API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_POINTER

Bad or NULL pointer.

CWBRC_INVALID_SYSTEM_HANDLE

Invalid system handle.

CWBRC_REJECTED_USER_EXIT

Command rejected by user exit program.

CWBRC_USR_EXIT_ERROR

Error in user exit program.

CWBRC_COMMAND_FAILED

Command failed.

CWBRC_COMMAND_TOO_LONG

Command string is too long.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: None

cwbRC_SetLibName

Purpose: Set the name of the library for this program object.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_SetLibName(  
    cwbRC_PgmHandle    program,  
    const char         *libraryName);
```

Parameters:

cwbRC_PgmHandle program - input

Handle that was returned by a previous call to the `cwbRC_CreatePgm` API. It identifies the program object.

const char *libraryName - input

Pointer to an ASCII string that contains the name of the library where the program resides.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWBRC_LIBRARY_NAME

Library name is too long.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: Use this function to change the name of the library that contains the program you want to call. This function should not be used to call a different program with different parameters.

cwbRC_SetParm

Purpose: Set the parameter value identified by the index. The index will range from 0 to the total number of parameters - 1. This number can be obtained by calling the `cwbRC_GetParmCount` API. Note that this function is to be used to change a parameter. Use `cwbRC_AddParm` to create the parameter.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_SetParm(  
    cwbRC_PgmHandle    program,  
    unsigned short     index,  
    unsigned short     type,  
    unsigned long      length,  
    const unsigned char *parameter);
```

Parameters:

cwbRC_PgmHandle handle - input

Handle that was returned by a previous call to the `cwbRC_CreatePgm` API. It identifies the program object.

unsigned short index - input

The number of the specific parameter in this program that should be changed. This index is zero-based.

unsigned short type - input

The type of parameter this is. Use one of the defined parameter types:

CWBRC_INPUT
CWBRC_OUTPUT
CWBRC_INOUT

If you want to automatically convert between local CCSID and host CCSID, add the appropriate convert flag to this field with a bitwise-OR. Use one of the defined parameter types:

CWBRC_TEXT_CONVERT
CWBRC_TEXT_CONVERT_INPUT
CWBRC_TEXT_CONVERT_OUTPUT

The latter two are intended for use with `CWBRC_INOUT` when conversion is only needed in one direction.

unsigned long length - input

The length of the parameter. If this is an `CWBRC_OUT` parameter, the length should be the length of the buffer where the returned parameter will be written.

const unsigned char * parameter - input

Pointer to a buffer that will contain the value if the type is `CWBRC_INPUT` or `CWBRC_INOUT`, or the place where the return parameter is to be written if the type is `CWBRC_OUTPUT` or `CWBRC_INOUT`.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWBRC_INVALID_TYPE

Invalid type specified.

CWBRC_INVALID_PARM_LENGTH

Invalid parameter length.

CWBRC_INVALID_PARM

Invalid parameter.

Usage: Parameter data is assumed to be binary. No conversion will be performed on the parameter data unless one of the conversion flags is set. For example:

```
cwbRC_SetParam( hPgm,  
                CWBRC_INOUT | CWBRC_TEXT_CONVERT_OUTPUT,  
                bufferSize,  
                buffer );
```

will use the buffer as is to send to the host, and will convert the output (for example, to ASCII) before putting the result into the buffer.

cwbRC_SetPgmName

Purpose: Set the name of the program for this program object.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_SetPgmName(  
    cwbRC_PgmHandle    program,  
    const char         *programName);
```

Parameters:

cwbRC_PgmHandle program - input

Handle that was returned by a previous call to the cwbRC_CreatePgm API. It identifies the program object.

const char *programName - input

Pointer to an ASCII string that contains the name of the program that you want to call.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWBRC_INVALID_PROGRAM

Invalid program handle.

CWBRC_PROGRAM_NAME

Program name is too long.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: Use this function to change the name of the program that you want to call. This function should not be used to change the program object to call a different program with different parameters.

cwbRC_StartSysEx

Purpose: This function starts a conversation with the specified system. If the conversation is successfully started, a handle is returned. Use this handle with all subsequent calls to issue commands or call programs. When the conversation no longer is needed, use the handle with the `cwbRC_StopSys` API to end the conversation. The `cwbRC_StartSysEx` API may be called multiple times within an application. If the same system object handle is used on `StartSysEx` calls, only one conversation with the iSeries server will be started. If you want multiple conversations to be active, you must call `StartSysEx` multiple times, specifying different system object handles.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_StartSysEx(  
    const cwbCO_SysHandle systemObj,  
    cwbRC_SysHandle *request);
```

Parameters:

const cwbCO_SysHandle systemObj - input

Handle to an existing system object of the system on which you want programs and commands to be run.

cwbRC_SysHandle *request - output

Pointer to a `cwbRC_SysHandle` where the handle of the command request will be returned.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWB_COMMUNICATIONS_ERROR

A communications error occurred.

CWB_SERVER_PROGRAM_NOT_FOUND

iSeries application not found.

CWB_HOST_NOT_FOUND

iSeries system inactive or does not exist.

CWB_SECURITY_ERROR

A security error has occurred.

CWB_LICENSE_ERROR

A license error has occurred.

CWB_CONFIG_ERROR

A configuration error has occurred.

CWBRC_SYSTEM_NAME

System name is too long.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory; may have failed to allocate temporary buffer.

CWB_NON_REPRESENTABLE_UNICODE_CHAR

One or more input Unicode characters have no representation in the codepage being used.

CWB_API_ERROR

General API failure.

Usage: None.

cwbRC_StopSys

Purpose: This function stops a conversation with the system specified by the handle. This handle can no longer be used to issue program calls or commands.

Syntax:

```
unsigned int CWB_ENTRY cwbRC_StopSys(  
    cwbRC_SysHandle    system);
```

Parameters:

cwbRC_SysHandle system - input

Handle that was returned by a previous call to the `cwbRC_StartSysEx` function. It identifies the iSeries system.

Return Codes: The following list shows common return values:

CWB_OK

Successful completion.

CWBRC_INVALID_SYSTEM_HANDLE

Invalid system handle.

Usage: None

Example: Using Remote iSeries Access for Windows Command/Distributed Program Call APIs

```
#ifdef UNICODE
    #define _UNICODE
#endif
#include <windows.h>

// Include the necessary RC/DPC Classes
#include <stdlib.h>
#include <iostream.h>
#include <TCHAR.H>
#include "cwbrc.h"
#include "cwbcosys.h"
/*****/

void main()
{
    cwbcO_SysHandle system;
    cwbrC_SysHandle request;
    cwbrC_PgmHandle program;

    // Create the system object
    if ( (cwbcO_CreateSystem("AS/400SystemName",&system)) != CWB_OK )
        return;

    // Start the system
    if ( (cwbrC_StartSysEx(system,&request)) != CWB_OK )
        return;

    // Call the command to create a library
    char* cmd1 = "CRTLIB LIB(RCTESTLIB) TEXT('RC TEST LIBRARY')";
    if ( (cwbrC_RunCmd(request, cmd1, 0)) != CWB_OK )
        return;

    cout << "Created Library" << endl;

    // Call the command to delete a library
    char* cmd2 = "DLTLIB LIB(RCTESTLIB)";
    if ( (cwbrC_RunCmd(request, cmd2, 0)) != CWB_OK )
        return;

    cout << "Deleted Library" << endl;

    // Create a program object to create a user space
    if ( cwbrC_CreatePgm(_TEXT("QUSCRTUS"),
                        _TEXT("QSYS"),
                        &program) != CWB_OK )
        return;

    // Add the parameters
    // name is DPCTESTSPC/QGPL
    unsigned char name[20] = {0xC4,0xD7,0xC3,0xE3,0xC5,0xE2,0xE3,0xE2,0xD7,0xC3,
                             0xD8,0xC7,0xD7,0xD3,0x40,0x40,0x40,0x40,0x40,0x40};

    // extended attribute is not needed
    unsigned char attr[10] = {0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40};

    // initial size is 100 bytes
    unsigned long size = 0x64000000;
}
```

```

    // initial value is blank
    unsigned char init = 0x40;

    // public authority is CHANGE
    unsigned char auth[10] = {0x5C,0xC3,0xC8,0xC1,0xD5,0xC7,0xC5,0x40,0x40,0x40};

    // description is DPC TEMP SPACE
    unsigned char desc[50] = {0xC4,0xD7,0xC3,0x40,0xE3,0xC5,0xD4,0xD7,0x40,0xE2,
                             0xD7,0xC1,0xC3,0xC5,0x40,0x40,0x40,0x40,0x40,0x40,
                             0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
                             0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,
                             0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x40};

    if ( cwbRC_AddParm(program, CWBRC_INPUT, 20, name) != CWB_OK)
        return;

    if ( cwbRC_AddParm(program, CWBRC_INPUT, 10, attr) != CWB_OK)
        return;

    if ( cwbRC_AddParm(program, CWBRC_INPUT, 4, (unsigned char*)&size) != CWB_OK)
        return;

    if ( cwbRC_AddParm(program, CWBRC_INPUT, 1, &init) != CWB_OK)
        return;

    if ( cwbRC_AddParm(program, CWBRC_INPUT, 10, auth) != CWB_OK)
        return;

    if ( cwbRC_AddParm(program, CWBRC_INPUT, 50, desc) != CWB_OK)
        return;

    // Call the program
    if ( cwbRC_CallPgm(request, program, 0) != CWB_OK )
        return;

    cout << "Created User Space" << endl;

    // Delete the program
    if ( cwbRC_DeletePgm(program) != CWB_OK )
        return;

    // Create a program object to delete a user space
    if ( cwbRC_CreatePgm(_TEXT("QUSDLTUS"),
                       _TEXT("QSYS"),
                       &program) != CWB_OK )
        return;

    // Add the parameters
    // error code structure will not be used
    unsigned long err = 0x00000000;

    if ( cwbRC_AddParm(program, CWBRC_INPUT, 20, name) != CWB_OK)
        return;

    if ( cwbRC_AddParm(program, CWBRC_INOUT, 4, (unsigned char*)&err) != CWB_OK)
        return;

    // Call the program

```

```

if ( cwbRC_CallPgm(request, program, 0) != CWB_OK )
    return;

// Delete the program
if ( cwbRC_DeletePgm(program) != CWB_OK )
    return;

cout << "Deleted User Space" << endl;

// Stop the system
if ( cwbRC_StopSys(request) != CWB_OK )
    return;

// Delete the system object
if ( cwbCO_DeleteSystem(system) != CWB_OK )
    return;
}

```

iSeries Access for Windows Serviceability APIs

The iSeries Access for Windows Serviceability application programming interfaces (APIs) allow you to log service file messages and events within your program. A set of APIs allows you to read the records from the service files that are created. These APIs allow you to write a customized service-file browser.

The following general categories of iSeries Access for Windows Serviceability API functions are provided:

- Writing message text to the History log
- Writing Trace entries to the Trace file
- Reading service files
- Retrieving message text that is associated with error handles

Why you should use iSeries Access for Windows Serviceability APIs:

The iSeries Access for Windows Serviceability APIs provide an efficient means of adding message logging and trace points to your code. Incorporate these functions into programs that are shipped as part of your product, and use them to help debug programs that are under development. The file structure supports multiple programs (that are identified by unique product and component strings) logging to the same files simultaneously. This provides a complete picture of logging activity on the client workstation.

iSeries Access for Windows Serviceability APIs required files:

Header file	Import library	Dynamic Link Library
cwbsv.h	cwbapi.lib	cwbsv.dll

Programmer's Toolkit:

The Programmer's Toolkit provides Serviceability documentation, access to the cwbsv.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select **Error Handling** → **C/C++ APIs**.

iSeries Access for Windows Serviceability APIs topics:

- "History log and trace files" on page 423
- "Error handles" on page 424
- "Typical use of Serviceability APIs" on page 424
- **iSeries Access for Windows Serviceability APIs listing**
- "Example: Using iSeries Access for Windows Serviceability APIs" on page 490
- "Serviceability APIs return codes" on page 30

History log and trace files

History log:

The log functions allow you to write message text the iSeries Access for Windows History Log. The message text needs to be displayable ASCII character data.

iSeries Access for Windows has instrumented all of its programs to log messages to the iSeries Access for Windows History Log. Messages also are logged by the DLLs that are supplied with the product.

The History Log is a file where message text strings are logged through the **cwbSV_LogMessageText** API. The log provides a history of activity that has taken place on the client workstation.

Trace files:

The trace functions allow you to log low-level events that occur as your program runs. For example, you want to track various return codes that were received from calling other functions. If your program is sending and receiving data, you may want to log the significant fields of the data (for example, function byte or bytes, and data length) to aid in debugging if something goes wrong. Use the **Detailed data trace** function (**cwbSV_LogTraceData**) to accomplish this.

Another form of trace, the **Entry Point trace** function, allows you to track entry into and exit from your routines. iSeries Access for Windows defines two different types of entry point trace points:

API trace point:

Use the API (application programming interface) trace point to track entry and exit from routines that you externalize to other programs.

SPI trace point:

Use the SPI (system programming interface) trace point to track entry and exit from key internal routines of the program that you want to trace.

The key piece of information that is provided on the APIs is a one-byte eventID. It allows you to identify which API or SPI is being entered or exited. Data such as input values can be traced on entry, as well as tracing output values on exit from a routine. These trace functions are intended to be used in pairs (for example, **cwbSV_LogAPIEntry** and **cwbSV_LogAPIExit**) in the routines that utilize them. These types of trace points provide a flow of control through the code.

iSeries Access for Windows has instrumented the procedural APIs described in this topic with Entry/Exit API trace points. When one of these procedural APIs is called, entry and exit trace points are logged to the Entry Point trace file if tracing is active. The Entry/Exit SPI trace logs internal calling sequences. The Detailed data trace function logs data which is useful in debugging problems.

iSeries Access for Windows supports the following types of traces:

Detailed (Data):

Allows you to trace a buffer of information at a point in your code via the **cwbSV_LogTraceData** API. This buffer can be a mixture of ASCII and/or binary values (for example, C-struct). The data is logged in binary form.

Entry/Exit (API):

A specialized form of trace which allows you to trace entry into and exit from your externalized routines via the **cwbSV_LogAPIEntry** and **cwbSV_LogAPIExit** APIs.

Entry/Exit (SPI):

A specialized form of trace that allows you to trace entry into and exit from your key internal routines by using the **cwbSV_LogSPIEntry** and **cwbSV_LogSPIExit** APIs.

Error handles

The error handle functions allow you to create an error handle (cwbSV_CreateErrHandle) to use on iSeries Access for Windows APIs that support it. If an error occurs (a non-zero return code) on the iSeries Access for Windows API call, you can call other error handle functions to retrieve information such as:

- The number of error messages (cwbSV_GetErrCount) that are associated with the return code
- The message text (cwbSV_GetErrMsgIndexed) for each of the error messages

Typical use of Serviceability APIs

History log:

Serviceability APIs provide a tracking mechanism for activity that is taking place on the client workstation. As a result, you can use the message-logging APIs to log messages to the iSeries Access for Windows History Log. Examples of messages to log include an indication that your application was started, and other significant events. For example, a log message may indicate that a file successfully was transferred to the iSeries server, a database query failed for some reason, or that a job was submitted for printing.

The product and component strings that you provide when you are using the Serviceability APIs allow your messages and events to be distinguished from other entries in the service files. The recommended hierarchy is to define a product ID, with one or many component IDs defined under it.

Error handles:

Use the error-handle parameter on iSeries Access for Windows C/C++ APIs to retrieve message text that is associated with a failure return code. This enables your application to display the message text, instead of providing your own text for the set of iSeries Access return codes.

iSeries Access for Windows Serviceability APIs listing

Note: Distinguish between API & SPI trace points:

- Definitions:
- API (Application Programming Interface)
 - SPI (System Programming Interface)

The recommended convention is that API entry/exit trace points should be put in routines that you externalize (export) to your users. Use SPI entry/exit trace points in key internal (non-exported) routines that you want to trace.

The following iSeries Access for Windows Serviceability APIs are listed alphabetically, and are grouped according to function:

Function	iSeries Access for Windows Serviceability APIs
Writing message text to a history log	cwbSV_CreateMessageTextHandle cwbSV_DeleteMessageTextHandle cwbSV_LogMessageText cwbSV_SetMessageComponent cwbSV_SetMessageProduct cwbSV_SetMessageClass
Writing trace data to a detail trace file	cwbSV_CreateTraceDataHandle cwbSV_DeleteTraceDataHandle cwbSV_LogTraceData cwbSV_SetTraceComponent cwbSV_SetTraceProduct

Function	iSeries Access for WindowsServiceability APIs
Writing trace points to an entry/exit trace file	cwbSV_CreateTraceAPIHandle cwbSV_CreateTraceSPIHandle cwbSV_DeleteTraceAPIHandle cwbSV_DeleteTraceSPIHandle cwbSV_LogAPIEntry cwbSV_LogAPIExit cwbSV_LogSPIEntry cwbSV_LogSPIExit cwbSV_SetAPIComponent cwbSV_SetAPIProduct cwbSV_SetSPIComponent cwbSV_SetSPIProduct
Reading service files	cwbSV_ClearServiceFile cwbSV_CloseServiceFile cwbSV_GetMaxRecordSize cwbSV_GetRecordCount cwbSV_GetServiceFileName cwbSV_OpenServiceFile
Reading service file records	cwbSV_CreateServiceRecHandle cwbSV_DeleteServiceRecHandle cwbSV_ReadNewestRecord cwbSV_ReadNextRecord cwbSV_ReadOldestRecord cwbSV_ReadPrevRecord
Reading service record header information	cwbSV_GetComponent cwbSV_GetDateStamp cwbSV_GetProduct cwbSV_GetServiceType cwbSV_GetTimeStamp
Reading history log service records	cwbSV_GetMessageText
Reading detail trace file service records	cwbSV_GetTraceData
Reading entry/exit trace file service records	cwbSV_GetTraceAPIData cwbSV_GetTraceAPIID cwbSV_GetTraceAPIType cwbSV_GetTraceSPIData cwbSV_GetTraceSPIID cwbSV_GetTraceSPIType
Retrieving message text associated with error handles	cwbSV_CreateErrHandle cwbSV_DeleteErrHandle cwbSV_GetErrClass cwbSV_GetErrClassIndexed cwbSV_GetErrCount cwbSV_GetErrFileName cwbSV_GetErrFileNameIndexed cwbSV_GetErrLibName cwbSV_GetErrLibNameIndexed cwbSV_GetErrSubstText cwbSV_GetErrSubstTextIndexed cwbSV_GetErrText cwbSV_GetErrTextIndexed

cwbSV_ClearServiceFile

Purpose: Clears the service file that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_ClearServiceFile(  
    cwbSV_ServiceFileHandle serviceFile,  
    cwbSV_ErrHandle         errorHandle);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the cwbSV_OpenServiceFile() function.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_FILE_IO_ERROR

File could not be cleared.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: None

cwbSV_CloseServiceFile

Purpose: Closes the service file identified by the handle provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_CloseServiceFile(  
    cwbSV_ServiceFileHandle serviceFile,  
    cwbSV_ErrHandle         errorHandle);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the cwbSV_OpenServiceFile() function.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_FILE_IO_ERROR

File could not be closed.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: None

cwbSV_CreateErrHandle

Purpose: This function creates an error message object and returns a handle to it. This error handle can be passed to iSeries Access for Windows APIs that support it. If an error occurs on one of these APIs, the error handle can be used to retrieve the error messages text that is associated with the API error.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_CreateErrHandle(  
    cwbSV_ErrHandle *errorHandle);
```

Parameters:

cwbSV_ErrHandle *errorHandle - input/output

Pointer to a cwbSV_ErrHandle where the handle will be returned.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed as handle address.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

Usage: None

cwbSV_CreateMessageTextHandle

Purpose: This function creates a message text object and returns a handle to it. This message handle can be used in your program to write message text to the currently active history log. The message text is supplied in a buffer passed on the cwbSV_LogMessageText() call.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_CreateMessageTextHandle(  
    char                *productID,  
    char                *componentID,  
    cwbSV_MessageTextHandle *messageTextHandle);
```

Parameters:

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this message entry. Parameter is optional, if null, no productID is set. NOTE: A maximum of CWBSV_MAX_PRODUCT_ID characters will be logged for the product ID. Larger strings will be truncated.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this message entry. Parameter is optional, if null, no componentID is set. NOTE: A maximum of CWBSV_MAX_COMP_ID characters will be logged for the component ID. Larger strings will be truncated.

cwbSV_MessageTextHandle * messageTextHandle - input/output

Pointer to a cwbSV_MessageTextHandle where the handle will be returned. This handle should be used in subsequent calls to the message text functions.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

Usage: It is recommended that you set a unique product ID and component ID in the message handle before using it to log message text. These ID's will distinguish your messages from other messages in the history log.

cwbSV_CreateServiceRecHandle

Purpose: This function creates a service record object and returns a handle to it.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_CreateServiceRecHandle(  
    cwbSV_ServiceRecHandle *serviceRecHandle);
```

Parameters:

cwbSV_ServiceRecHandle * serviceRecHandle - input/output

Pointer to a cwbSV_ServiceRecordHandle where the handle will be returned. This handle should be used in subsequent calls to the service record functions.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed as handle address.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

Usage: This handle can be used in your program to read records from an open service file and extract information from the record.

cwbSV_CreateTraceAPIHandle

Purpose: This function creates a trace API object and returns a handle to it. This trace API handle can be used in your program to log entry to and exit from your API entry points.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_CreateTraceAPIHandle(  
    char          *productID,  
    char          *componentID,  
    cwbSV_TraceAPIHandle *traceAPIHandle);
```

Parameters:

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this message entry. Parameter is optional, if null, no productID is set. NOTE: A maximum of CWBSV_MAX_PRODUCT_ID characters will be logged for the product ID. Larger strings will be truncated.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this message entry. Parameter is optional, if null, no componentID is set. NOTE: A maximum of CWBSV_MAX_COMP_ID characters will be logged for the component ID. Larger strings will be truncated.

cwbSV_TraceAPIHandle * traceAPIHandle - input/output

Pointer to a cwbSV_TraceAPIHandle where the handle will be returned. This handle should be used in subsequent calls to the trace API functions.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

Usage: It is recommended that you set a unique product ID and component ID in the trace data handle before using it to log trace entries. These ID's will distinguish your trace entries from other entries in the trace file.

cwbSV_CreateTraceDataHandle

Purpose: This function creates a trace data object and returns a handle to it. This trace handle can be used in your program to log trace information to trace files. The trace information is supplied in a buffer passed on `cwbSV_LogTraceData()` calls.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_CreateTraceDataHandle(  
    char                *productID,  
    char                *componentID,  
    cwbSV_TraceDataHandle *traceDataHandle);
```

Parameters:

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this message entry. Parameter is optional, if null, no productID is set. NOTE: A maximum of `CWBSV_MAX_PRODUCT_ID` characters will be logged for the product ID. Larger strings will be truncated.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this message entry. Parameter is optional, if null, no componentID is set. NOTE: A maximum of `CWBSV_MAX_COMP_ID` characters will be logged for the component ID. Larger strings will be truncated.

cwbSV_TraceDataHandle * traceDataHandle - input/output

Pointer to a `cwbSV_TraceDataHandle` where the handle will be returned. This handle should be used in subsequent calls to the trace data functions.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

Usage: It is recommended that you set a unique product ID and component ID in the trace data handle before using it to log trace entries. These ID's will distinguish your trace entries from other entries in the trace file.

cwbSV_CreateTraceSPIHandle

Purpose: This function creates a trace SPI object and returns a handle to it. This trace SPI handle can be used in your program to log entry to and exit from your SPI entry points.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_CreateTraceSPIHandle(  
    char          *productID,  
    char          *componentID,  
    cwbSV_TraceSPIHandle *traceSPIHandle);
```

Parameters:

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this message entry. Parameter is optional, if null, no productID is set. NOTE: A maximum of CWBSV_MAX_PRODUCT_ID characters will be logged for the product ID. Larger strings will be truncated.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this message entry. Parameter is optional, if null, no componentID is set. NOTE: A maximum of CWBSV_MAX_COMP_ID characters will be logged for the component ID. Larger strings will be truncated.

cwbSV_TraceSPIHandle * traceSPIHandle - input/output

Pointer to a cwbSV_TraceSPIHandle where the handle will be returned. This handle should be used in subsequent calls to the trace SPI functions.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

Usage: It is recommended that you set a unique product ID and component ID in the trace data handle before using it to log trace entries. These ID's will distinguish your trace entries from other entries in the trace file.

cwbSV_DeleteErrHandle

Purpose: This function deletes the error message object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_DeleteErrHandle(  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbSV_ErrHandle errorHandle - output

Handle that was returned by a previous call to the cwbSV_CreateErrHandle() function.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should be made when the handle is no longer needed.

cwbSV_DeleteMessageTextHandle

Purpose: This function deletes the message text object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_DeleteMessageTextHandle(  
    cwbSV_MessageTextHandle messageTextHandle);
```

Parameters:

cwbSV_MessageTextHandle messageTextHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateMessageTextHandle()` function.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Unusable handle passed in on request.

Usage: This call should be made when the handle is no longer needed.

cwbSV_DeleteServiceRecHandle

Purpose: This function deletes the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_DeleteServiceRecHandle(  
    cwbSV_ServiceRecHandle serviceRecHandle);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle()` function.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should be made when the handle is no longer needed.

cwbSV_DeleteTraceAPIHandle

Purpose: This function deletes the trace API object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_DeleteTraceAPIHandle(  
    cwbSV_TraceAPIHandle traceAPIHandle);
```

Parameters:

cwbSV_TraceAPIHandle traceAPIHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateTraceAPIHandle()` function.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should be made when the handle is no longer needed.

cwbSV_DeleteTraceDataHandle

Purpose: This function deletes the trace data object that is identified by the trace handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_DeleteTraceDataHandle(  
    cwbSV_TraceDataHandle traceDataHandle);
```

Parameters:

cwbSV_TraceDataHandle traceDataHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateTraceDataHandle()` function.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should be made when the handle is no longer needed.

cwbSV_DeleteTraceSPIHandle

Purpose: This function deletes the trace SPI object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_DeleteTraceSPIHandle(  
    cwbSV_TraceSPIHandle traceSPIHandle);
```

Parameters:

cwbSV_TraceSPIHandle traceSPIHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateTraceSPIHandle()` function.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should be made when the handle is no longer needed.

cwbSV_GetComponent

Purpose: Returns the component ID value for the service record object that is identified by the handle provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetComponent(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *componentID,  
    unsigned long componentIDLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle` function.

char * componentID - input/output

Pointer to a buffer that will receive the component ID that is stored in the record that is identified by the handle.

unsigned long componentIDLength - input

Length of the receive buffer passed in. It should include space for the ending null character. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set. NOTE: The recommended size is `CWBSV_MAX_COMP_ID`.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: The service record handle needs to be filled in by a call to a "read" function before calling this routine, otherwise a NULL string will be returned. This function is valid for all service record types.

cwbSV_GetDateStamp

Purpose: Returns the date stamp (in localized format) for the service record that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetDateStamp(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *dateStamp,  
    unsigned long dateStampLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the cwbSV_CreateServiceRecHandle function.

char * dateStamp - input/output

Pointer to a buffer that will receive the datestamp that is stored in the record that is identified by the handle.

unsigned long dateStampLength - input

Length of the receive buffer passed in. It should include space for the ending null character. If the buffer is too small, the value will be truncated, and CWB_BUFFER_OVERFLOW and returnLength will be set. NOTE: The recommended size is CWBSV_MAX_DATE_VALUE.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: The service record handle needs to be filled in by a call to a "read" function before calling this routine, otherwise a NULL string will be returned. This function is valid for all service record types.

cwbSV_GetErrClass

Purpose: Returns the message class associated with the top-level (most recent) error that is identified by the error handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrClass(  
    cwbSV_ErrHandle errorHandle,  
    unsigned long *errorClass);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the cwbSV_CreateErrHandle() function.

unsigned long * errorClass - output

Pointer to a variable that will receive the error class that is stored in the error that is identified by the handle.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_NO_ERROR_MESSAGES

No error messages associated with error handle.

Usage: None

cwbSV_GetErrClassIndexed

Purpose: Returns the message class associated with the error index provided. An index value of 1 will retrieve the lowest-level (for example, the oldest) message that is associated with the error handle. An index value of "cwbSV_GetErrCount()'s returned errorCount" will retrieve the top-level (for example, the most recent) message associated with the error handle.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrClassIndexed(  
    cwbSV_ErrHandle  errorHandle,  
    unsigned long    errorIndex,  
    unsigned long    *errorClass);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the cwbSV_CreateErrHandle() function.

unsigned long errorIndex - input

Index value that indicates which error text to return if multiple errors are associated with the error handle.

unsigned long * errorClass - output

Pointer to a variable that will receive the error class that is stored in the error that is identified by the index.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_NO_ERROR_MESSAGES

No error messages associated with error handle.

Usage: Valid index values are from 1 to cwbSV_GetErrCount()'s return value. Index values less than 1 act as if 1 was passed. Index values greater than cwbSV_GetErrCount() act as if errorCount was passed.

cwbSV_GetErrCount

Purpose: Returns the number of messages associated with the error handle provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrCount(  
    cwbSV_ErrHandle errorHandle,  
    unsigned long *errorCount);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the cwbSV_CreateErrHandle() function.

unsigned long * errorCount - input/output

Pointer to variable that receives the number of messages associated with this error handle. If zero is returned, no errors are associated with the error handle.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: None

cwbSV_GetErrFileName

Purpose: Returns the message file name for the top-level (the most recent) message added to the error handle provided. This message attribute only pertains to messages returned from the iSeries server. The file name is the name of the iSeries server message file that contains the message.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrFileName(  
    cwbSV_ErrHandle  errorHandle,  
    char              *fileName,  
    unsigned long    fileNameLength,  
    unsigned long    *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` API.

char * fileName - input/output

Pointer to a buffer that will receive the message file name stored in the error identified by the handle. The value returned is an ASCII string.

unsigned long fileNameLength - input

Length of the receive buffer passed in. It should include space for the terminating null character. If the buffer is too small, the value will be truncated and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set. NOTE: The recommended size is `CWBSV_MAX_MSGFILE_NAME`.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Invalid handle.

CWBSV_NO_ERROR_MESSAGES

No messages are in the error handle.

CWBSV_ATTRIBUTE_NOT_SET

Attribute not set in current message.

Usage: iSeries server messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` API's. In these cases, you can use this API to retrieve the message file name for the iSeries server messages contained in the error handle. If there is no message file name attribute for the message, return code `CWBSV_ATTRIBUTE_NOT_SET` will be returned.

cwbSV_GetErrFileNameIndexed

Purpose: Returns the message file name for the message identified by the index provided. This message attribute only pertains to messages returned from the iSeries server. The file name is the name of the iSeries server message file containing the message.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrFileNameIndexed(  
    cwbSV_ErrHandle  errorHandle,  
    unsigned long    index,  
    char             *fileName,  
    unsigned long    fileNameLength,  
    unsigned long    *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` API.

unsigned long index - input

Index value indicating which message file name to return if multiple errors are associated with the error handle. The valid index range is from 1 to the number of messages contained in the error handle. The number of messages can be obtained by calling the `cwbSV_GetErrCount()` API.

char * fileName - input/output

Pointer to a buffer that will receive the message file name stored in the error identified by the index. The value returned is an ASCII string.

unsigned long fileNameLength - input

Length of the receive buffer passed in. It should include space for the terminating null character. If the buffer is too small, the value will be truncated and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set. NOTE: The recommended size is `CWBSV_MAX_MSGFILE_NAME`.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Invalid handle.

CWBSV_NO_ERROR_MESSAGES

No messages are in the error handle.

CWBSV_ATTRIBUTE_NOT_SET

Attribute not set in current message.

Usage: iSeries server messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` API's. In these cases, you can use this API to retrieve the message file name for the iSeries server messages contained in the error handle. If there is no message file name attribute for the message, return code `CWBSV_ATTRIBUTE_NOT_SET` will be returned. An index value of 1 works with the lowest-level (i.e. oldest) message in the error handle. An index value equal to the count returned

by the `cwbSV_GetErrCount()` API works with the top-level (i.e. most recent) message in the error handle. Index values less than 1 act as if 1 was passed in. Index values greater than the number of messages contained in the error handle act as if the returned count value from the `cwbSV_GetErrCount()` API was passed in.

cwbSV_GetErrLibName

Purpose: Returns the message file library name for the top-level (i.e. most recent) message added to the error handle provided. This message attribute only pertains to messages returned from the iSeries server. The library name is the name of the iSeries library containing the message file for the message.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrLibName(  
    cwbSV_ErrHandle  errorHandle,  
    char             *libraryName,  
    unsigned long    libraryNameLength,  
    unsigned long    *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` API.

char * libraryName - input/output

Pointer to a buffer that will receive the message file library name stored in the error identified by the handle. The value returned is an ASCII string.

unsigned long libraryNameLength - input

Length of the receive buffer passed in. It should include space for the terminating null character. If the buffer is too small, the value will be truncated and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set. NOTE: The recommended size is `CWBSV_MAX_MSGFILE_LIBR`.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Invalid handle.

CWBSV_NO_ERROR_MESSAGES

No messages are in the error handle.

CWBSV_ATTRIBUTE_NOT_SET

Attribute not set in current message.

Usage: iSeries messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` API's. In these cases, you can use this API to retrieve the message file library name for the iSeries messages contained in the error handle. If there is no message file library name attribute for the message, return code `CWBSV_ATTRIBUTE_NOT_SET` will be returned.

cwbSV_GetErrLibNameIndexed

Purpose: Returns the message file library name for the message identified by the index provided. This message attribute only pertains to messages returned from the iSeries server. The library name is the name of the iSeries library containing the message file for the message.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrLibNameIndexed(  
    cwbSV_ErrHandle  errorHandle,  
    unsigned long    index,  
    char             *libraryName,  
    unsigned long    libraryNameLength,  
    unsigned long    *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` API.

unsigned long index - input

Index value indicating which message file library name to return if multiple errors are associated with the error handle. The valid index range is from 1 to the number of messages contained in the error handle. The number of messages can be obtained by calling the `cwbSV_GetErrCount()` API.

char * libraryName - input/output

Pointer to a buffer that will receive the message file library name stored in the error identified by the index. The value returned is an ASCII string.

unsigned long libraryNameLength - input

Length of the receive buffer passed in. It should include space for the terminating null character. If the buffer is too small, the value will be truncated and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set. NOTE: The recommended size is `CWBSV_MAX_MSGFILE_LIBR`.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Invalid handle.

CWBSV_NO_ERROR_MESSAGES

No messages are in the error handle.

CWBSV_ATTRIBUTE_NOT_SET

Attribute not set in current message.

Usage: iSeries messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` API's. In these cases, you can use this API to retrieve the message file library name for the iSeries messages contained in the error handle. If there is no message file library name attribute for the message, return code `CWBSV_ATTRIBUTE_NOT_SET` will be returned. An index value of 1 works with the lowest-level (i.e. oldest) message in the error handle. An index value equal to the count returned

by the `cwbSV_GetErrCount()` API works with the top-level (i.e. most recent) message in the error handle. Index values less than 1 act as if 1 was passed in. Index values greater than the number of messages contained in the error handle act as if the returned count value from the `cwbSV_GetErrCount()` API was passed in.

cwbSV_GetErrSubstText

Purpose: Returns the message substitution text for the top-level (the most recent) message identified by the error handle provided. This message attribute only pertains to messages returned from the iSeries server. The substitution text is the data inserted into the substitution variable fields defined for the message.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrSubstText(  
    cwbSV_ErrHandle  errorHandle,  
    char             *substitutionText,  
    unsigned long    substitutionTextLength,  
    unsigned long    *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` API.

char * substitutionText - input/output

Pointer to a buffer that will receive the substitution text for the message identified by the handle. NOTE: The data returned is binary, hence it is NOT returned as an ASCII string. Any character strings contained in the substitution text are returned as EBCDIC values.

unsigned long substitutionTextLength - input

Length of the receive buffer passed in. If the buffer is too small, the value will be truncated and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output data if the receive buffer is too small. It will also be set to the actual number of bytes of output data returned upon successful completion.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Invalid handle.

CWBSV_NO_ERROR_MESSAGES

No messages are in the error handle.

CWBSV_ATTRIBUTE_NOT_SET

Attribute not set in current message.

Usage: iSeries server messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` API's. In these cases, you can use this API to retrieve the substitution text for the iSeries server messages contained in the error handle. If there is no substitution text for the message, return code `CWBSV_ATTRIBUTE_NOT_SET` will be returned. Use the `returnLength` parameter to determine the actual number of bytes returned in the substitution text when the return code is `CWB_OK`. The substitution text returned on this API could be used on a subsequent host retrieve message API call (`QSYS/QMHRTVM`) to retrieve the format of the substitution text or to return secondary help text with the substitution text added in. Host API's are called using the `cwbRC_CallPgm()` API.

cwbSV_GetErrSubstTextIndexed

Purpose: Returns the message substitution text for the message identified by the index provided. This message attribute only pertains to messages returned from the iSeries server. The substitution text is the data inserted into the substitution variable fields defined for the message.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrSubstTextIndexed(  
    cwbSV_ErrHandle  errorHandle,  
    unsigned long    index,  
    char             *substitutionText,  
    unsigned long    substitutionTextLength,  
    unsigned long    *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` API.

unsigned long index - input

Index value indicating which substitution text to return if multiple errors are associated with the error handle. The valid index range is from 1 to the number of messages contained in the error handle. The number of messages can be obtained by calling the `cwbSV_GetErrCount()` API.

char * substitutionText - input/output

Pointer to a buffer that will receive the substitution text stored in the error identified by the index. Note: The data returned is binary, hence it is NOT returned as an ASCII string. Any character strings contained in the substitution text are returned as EBCDIC values.

unsigned long substitutionTextLength - input

Length of the receive buffer passed in. If the buffer is too small, the value will be truncated and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output data if the receive buffer is too small. It will also be set to the actual number of bytes of output data returned upon successful completion.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Invalid handle.

CWBSV_NO_ERROR_MESSAGES

No messages are in the error handle.

CWBSV_ATTRIBUTE_NOT_SET

Attribute not set in current message.

Usage: iSeries server messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` API's. In these cases, you can use this API to retrieve the substitution text for the iSeries server messages contained in the error handle. If there is no substitution text for the message, return code `CWBSV_ATTRIBUTE_NOT_SET` will be returned. An index value of 1 works with the

lowest-level (i.e. oldest) message in the error handle. An index value equal to the count returned by the `cwbSV_GetErrCount()` API works with the top-level (i.e. most recent) message in the error handle. Index values less than 1 act as if 1 was passed in. Index values greater than the number of messages contained in the error handle act as if the returned count value from the `cwbSV_GetErrCount()` API was passed in. Use the `returnLength` parameter to determine the actual number of bytes returned in the substitution text when the return code is `CWB_OK`. The substitution text returned on this API could be used on a subsequent host retrieve message API call (`QSYS/QMHRTVM`) to retrieve the format of the substitution text or to return secondary help text with the substitution text added in. Host API's are called using the `cwbRC_CallPgm()` API.

cwbSV_GetErrText

Purpose: Returns the message text associated with the top-level (for example, the most recent) error that is identified by the error handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrText(  
    cwbSV_ErrHandle  errorHandle,  
    char              *errorText,  
    unsigned long     errorTextLength,  
    unsigned long     *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` function.

char * errorText - input/output

Pointer to a buffer that will receive the error message text that is stored in the error that is identified by the handle.

unsigned long errorTextLength - input

Length of the receive buffer passed in. It should include space for the ending null character. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_NO_ERROR_MESSAGES

No error messages associated with error handle.

Usage: None

cwbSV_GetErrTextIndexed

Purpose: Returns the message text associated with the error index provided. An index value of 1 will retrieve the lowest-level (for example, the oldest) message that is associated with the error handle. An index value of "cwbSV_GetErrCount()'s returned errorCount" will retrieve the top-level (for example, the most recent) message associated with the error handle.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetErrTextIndexed(  
    cwbSV_ErrHandle  errorHandle,  
    unsigned long    errorIndex,  
    char             *errorText,  
    unsigned long    errorTextLength,  
    unsigned long    *returnLength);
```

Parameters:

cwbSV_ErrHandle errorHandle - input

Handle that was returned by a previous call to the cwbSV_CreateErrHandle() function.

unsigned long errorIndex - input

Index value that indicates which error text to return if multiple errors are associated with the error handle.

char * errorText - input/output

Pointer to a buffer that will receive the error message text that is stored in the error that is identified by the index.

unsigned long errorTextLength - input

Length of the receive buffer passed in. It should include space for the ending null character. If the buffer is too small, the value will be truncated, and CWB_BUFFER_OVERFLOW and returnLength will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_NO_ERROR_MESSAGES

No error messages associated with error handle.

Usage: Valid index values are from 1 to cwbSV_GetErrCount()'s return value. Index values less than 1 act as if 1 was passed. Index values greater than cwbSV_GetErrCount() act as if errorCount was passed.

cwbSV_GetMaxRecordSize

Purpose: Returns the size (in bytes) of the largest record in the service file that is identified by the file handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetMaxRecordSize(  
    cwbSV_ServiceFileHandle serviceFile,  
    unsigned long *maxRecordSize);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the cwbSV_OpenServiceFile function.

unsigned long * recordCount - input/output

Pointer to variable that receives the size of the largest record in the file.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: None

cwbSV_GetMessageText

Purpose: Returns the message text portion of the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetMessageText(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *messageText,  
    unsigned long messageTextLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle` function.

char * messageText - input/output

Pointer to a buffer that will receive the message text that is stored in the record that is identified by the handle.

unsigned long messageTextLength - input

Length of the receive buffer passed in. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output data if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_MESSAGE_REC`.

Usage: If the record type is not `CWBSV_MESSAGE_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_GetProduct

Purpose: Returns the product ID value for the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetProduct(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *productID,  
    unsigned long productIDLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle` function.

char * productID - input/output

Pointer to a buffer that will receive the product ID that is stored in the record that is identified by the handle.

unsigned long productIDLength - input

Length of the receive buffer passed in. It should include space for the ending null character. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set. NOTE: The recommended size is `CWBSV_MAX_PRODUCT_ID`.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: The service record handle needs to be filled in by a call to a "read" function before calling this routine, otherwise a NULL string will be returned. This function is valid for all service record types.

cwbSV_GetRecordCount

Purpose: Returns the total numbers of records in the service file that is identified by the file handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetRecordCount(  
    cwbSV_ServiceFileHandle serviceFile,  
    unsigned long            *recordCount);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the cwbSV_OpenServiceFile function.

unsigned long * recordCount - input/output

Pointer to variable that receives the total number of records in the file.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: None

cwbSV_GetServiceFileName

Purpose: Returns the fully-qualified path and file name of where the service records are being logged to for a particular file type.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetServiceFileName(  
    cwbSV_ServiceFileType serviceFileType,  
    char *fileName,  
    unsigned long fileNameLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceFileType serviceFileType - input

Value indicating which service file name you want returned. - CWBSV_HISTORY_LOG - CWBSV_PROBLEM_LOG - CWBSV_DETAIL_TRACE_FILE - CWBSV_ENTRY_EXIT_TRACE_FILE

char * fileName - input/output

Pointer to a buffer that will receive the service file name associated with the one that was requested.

unsigned long fileNameLength - input

Length of the receive buffer passed in. It should include space for the ending null character. If the buffer is too small, the value will be truncated, and CWB_BUFFER_OVERFLOW and returnLength will be set. NOTE: The recommended size is CWBSV_MAX_FILE_PATH.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWBSV_INVALID_FILE_TYPE

Unusable file type passed-in.

Usage: The filename string returned could be used as input to the cwbSV_OpenServiceFile() routine.

cwbSV_GetServiceType

Purpose: Returns the type of record (trace, message, entry/exit, and so forth) for the service record that is identified by the handle that is provided. Note: The service record needs to be filled in by a call to a "read" function before calling this function.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetServiceType(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    cwbSV_ServiceRecType *serviceType,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle` function.

cwbSV_ServiceRecType * serviceType - output

Pointer to a `cwbSV_ServiceRecType` where the `serviceType` will be returned. -
CWBSV_MESSAGE_REC - CWBSV_PROBLEM_REC - CWBSV_DATA_TRACE_REC -
CWBSV_API_TRACE_REC - CWBSV_SPI_TRACE_REC

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Unusable record type detected.

Usage: The service record handle needs to be filled in by a call to a "read" function before calling this routine, otherwise `CWBSV_INVALID_RECORD_TYPE` will be returned.

cwbSV_GetTimeStamp

Purpose: Returns the timestamp (in localized format) for the service record that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTimeStamp(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *timeStamp,  
    unsigned long timeStampLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle` function.

char * timeStamp - input/output

Pointer to a buffer that will receive the timestamp that is stored in the record that is identified by the handle.

unsigned long timeStampLength - input

Length of the receive buffer passed in. It should include space for the ending null character. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set. NOTE: The recommended size is `CWBSV_MAX_TIME_VALUE`.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output string if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: The service record handle needs to be filled in by a call to a "read" function before calling this routine, otherwise a NULL string will be returned. This function is valid for all service record types.

cwbSV_GetTraceAPIData

Purpose: Returns the API trace data portion of the service record that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTraceAPIData(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *apiData,  
    unsigned long apiDataLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle()` function.

char * apiData - input/output

Pointer to a buffer that will receive the API trace data that is stored in the record that is identified by the handle. Note: The data that is returned is binary. Hence, it is NOT returned as an ASCII string.

unsigned long apiDataLength - input

Length of the receive buffer passed in. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output data if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_API_TRACE_REC`.

Usage: If the record type is not `CWBSV_API_TRACE_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_GetTraceAPIID

Purpose: Returns the API event ID of the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTraceAPIID(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *apiID);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle()` function.

char * apiID - input/output

Pointer to one-byte field that receives the API event ID.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_API_TRACE_REC`.

Usage: If the record type is not `CWBSV_API_TRACE_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_GetTraceAPIType

Purpose: Returns the API event type of the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTraceAPIType(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    cwbSV_EventType *eventType,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle()` function.

cwbSV_EventType * eventType - output

Pointer to a `cwbSV_EventType` where the event type will be returned. - `CWBSV_ENTRY_POINT` - `CWBSV_EXIT_POINT`

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_API_TRACE_REC`.

CWBSV_INVALID_EVENT_TYPE

Unusable event type detected.

Usage: If the record type is not `CWBSV_API_TRACE_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_GetTraceData

Purpose: Returns the trace data portion of the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTraceData(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *traceData,  
    unsigned long traceDataLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle()` function.

char * traceData - input/output

Pointer to a buffer that will receive the trace data that is stored in the record that is identified by the handle. Note: The data that is returned is binary. Hence, it is NOT returned as an ASCII string.

unsigned long traceDataLength - input

Length of the receive buffer passed in. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output data if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_DATA_TRACE_REC`.

Usage: If the record type is not `CWBSV_TRACE_DATA_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_GetTraceSPIData

Purpose: Returns the SPI trace data portion of the service record that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTraceSPIData(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *spiData,  
    unsigned long spiDataLength,  
    unsigned long *returnLength);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle()` function.

char * spiData - input/output

Pointer to a buffer that will receive the SPI trace data that is stored in the record that is identified by the handle. Note: The data that is returned is binary. Hence, it is NOT returned as an ASCII string.

unsigned long spiDataLength - input

Length of the receive buffer passed in. If the buffer is too small, the value will be truncated, and `CWB_BUFFER_OVERFLOW` and `returnLength` will be set.

unsigned long * returnLength - input/output

Optional, may be NULL. A return address to store the number of bytes needed to hold the output data if the receive buffer is too small.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_BUFFER_OVERFLOW

Output buffer too small, data truncated.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_SPI_TRACE_REC`.

Usage: If the record type is not `CWBSV_SPI_TRACE_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_GetTraceSPIID

Purpose: Returns the SPI event ID of the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTraceSPIID(  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    char *spiID);
```

Parameters:

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle()` function.

char * spiID - input/output

Pointer to one-byte field that receives the SPI event ID.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_SPI_TRACE_REC`.

Usage: If the record type is not `CWBSV_SPI_TRACE_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_GetTraceSPIType

Purpose: Returns the SPI event type of the service record object that is identified by the handle that is provided.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_GetTraceSPIType(  
    cwbSV_ServiceRechandle serviceRechandle,  
    cwbSV_EventType *eventType,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbSV_ServiceRechandle serviceRechandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRechandle()` function.

cwbSV_EventType * eventType - output

Pointer to a `cwbSV_EventType` where the event type will be returned. - `CWBSV_ENTRY_POINT` - `CWBSV_EXIT_POINT`

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed on output parameter.

CWB_INVALID_HANDLE

Handle is not valid.

CWBSV_INVALID_RECORD_TYPE

Type is not `CWBSV_SPI_TRACE_REC`.

CWBSV_INVALID_EVENT_TYPE

Unusable event type detected.

Usage: If the record type is not `CWBSV_SPI_TRACE_REC`, a return code of `CWBSV_INVALID_RECORD_TYPE` will be returned. (note: `cwbSV_GetServiceType()` returns the current record type)

cwbSV_LogAPIEntry

Purpose: This function will log an API entry point to the currently active entry/exit trace file. The product and component ID's set in the entry will be written along with the date and time of the when the data was logged. The apiID, along with any optional data that is passed on the request, will also be logged.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_LogAPIEntry(  
    cwbSV_TraceAPIHandle traceAPIHandle,  
    unsigned char        apiID,  
    char                *apiData,  
    unsigned long        apiDataLength);
```

Parameters:

cwbSV_TraceAPIHandle traceAPIHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceAPIHandle()`.

unsigned char apiID - input

A unique one-character code that will distinguish this API trace point from others that are logged by your program. Definition of these codes are left up to the caller of this API. The recommended approach is to use the defined range (0x00 - 0xFF) for each unique component in your product (that is, start at 0x00 for each component)

char * apiData - input

Points to a buffer that contains additional data (for example, input parameter values from your caller) that you want to log along with this entry point. Parameter is optional, it is ignored if the address is NULL or the data length is zero. This buffer can contain binary data because the length parameter is used in determining the amount to trace.

unsigned long apiDataLength - input

Specifies the number of bytes in the API data buffer to log for this trace entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should allow be used in conjunction with a corresponding "`cwbSV_LogAPIExit()`". It is recommended that these calls would be put at the beginning and end of an API routine that you write. The other method would be to use these log functions around calls to external routines that are not written by you.

cwbSV_LogAPIExit

Purpose: This function will log an API exit point to the currently active entry/exit trace file. The product and component ID's set in the entry will be written along with the date and time of the when the data was logged. The API ID, along with any optional data that is passed on the request, will also be logged.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_LogAPIExit(  
    cwbSV_TraceAPIHandle traceAPIHandle,  
    unsigned char        apiID,  
    char                *apiData,  
    unsigned long        apiDataLength);
```

Parameters:

cwbSV_TraceAPIHandle traceAPIHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceAPIHandle()`.

unsigned char apiID - input

A unique one-character code that will distinguish this API trace point from others that are logged by your program. Definition of these codes are left up to the caller of this API. The recommended approach is to use the defined range (0x00 - 0xFF) for each unique component in your product (that is, start at 0x00 for each component)

char * apiData - input

Points to a buffer that contains additional data (for example, output parameter values passed back to your caller) that you want to log along with this exit point. Parameter is optional, it is ignored if the address is NULL or the data length is zero. This buffer can contain binary data because the length parameter is used in determining the amount to trace.

unsigned long apiDataLength - input

Specifies the number of bytes in the API data buffer to log for this trace entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should allow be used in conjunction with a corresponding "`cwbSV_LogAPIEntry()`". It is recommended that these calls would be put at the beginning and end of an API routine that you write. The other method would be to use these log functions around calls to external routines that are not written by you.

cwbSV_LogMessageText

Purpose: This function will log the supplied message text to the currently active history log. The product and component ID's set in the entry will be written along with the date and time of the when the text was logged.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_LogMessageText(  
    cwbSV_MessageTextHandle messageTextHandle,  
    char *messageText,  
    unsigned long messageTextLength);
```

Parameters:

cwbSV_MessageTextHandle messageTextHandle - input

Handle that was returned by a previous call to `cwbSV_CreateMessageTextHandle()`.

char * messageText - input

Points to a buffer that contains the message text you want to log.

unsigned long messageTextLength - input

Specifies the number of bytes in the message text buffer to log for this message entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Unusable handle passed in on request.

Usage: None

cwbSV_LogSPIEntry

Purpose: This function will log an SPI entry point to the currently active entry/exit trace file. The product and component ID's set in the entry will be written along with the date and time of the when the data was logged. The spiID, along with any optional data that is passed on the request, will also be logged.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_LogSPIEntry(  
    cwbSV_TraceSPIHandle traceSPIHandle,  
    unsigned char        spiID,  
    char                *spiData,  
    unsigned long        spiDataLength);
```

Parameters:

cwbSV_TraceSPIHandle traceSPIHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceSPIHandle()`.

unsigned char spiID - input

A unique one-character code that will distinguish this SPI trace point from others that are logged by your program. Definition of these codes are left up to the caller of this API. The recommended approach is to use the defined range (0x00 - 0xFF) for each unique component in your product (that is, start at 0x00 for each component)

char * spiData - input

Points to a buffer that contains additional data (for example, input parameter values from your caller) that you want to log along with this entry point. Parameter is optional, it is ignored if the address is NULL or the data length is zero. This buffer can contain binary data because the length parameter is used in determining the amount to trace.

unsigned long spiDataLength - input

Specifies the number of bytes in the SPI data buffer to log for this trace entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

IHandle is not valid.

Usage: This call should allow be used in conjunction with a corresponding "`cwbSV_LogSPIExit()`". It is recommended that these calls would be put at the beginning and end of an API routine that you write. The other method would be to use these log functions around calls to external routines that are not written by you.

cwbSV_LogSPIExit

Purpose: This function will log an SPI exit point to the currently active entry/exit trace file. The product and component ID's set in the entry will be written along with the date and time of the when the data was logged. The spiID, along with any optional data that is passed on the request, will also be logged.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_LogSPIExit(  
    cwbSV_TraceSPIHandle traceSPIHandle,  
    unsigned char        spiID,  
    char                *spiData,  
    unsigned long        spiDataLength);
```

Parameters:

cwbSV_TraceSPIHandle traceSPIHandle - input

Handle that was returned by a previous call to cwbSV_CreateTraceSPIHandle().

unsigned char spiID - input

A unique one-character code that will distinguish this SPI trace point from others that are logged by your program. Definition of these codes are left up to the caller of this API. The recommended approach is to use the defined range (0x00 - 0xFF) for each unique component in your product (that is, start at 0x00 for each component)

char * spiData - input

Points to a buffer that contains additional data (for example, output parameter values passed back to your caller) that you want to log along with this exit point. Parameter is optional, it is ignored if the address is NULL or the data length is zero. This buffer can contain binary data because the length parameter is used in determining the amount to trace.

unsigned long spiDataLength - input

Specifies the number of bytes in the SPI data buffer to log for this trace entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This call should allow be used in conjunction with a corresponding "cwbSV_LogSPIEntry()". It is recommended that these calls would be put at the beginning and end of an API routine that you write. The other method would be to use these log functions around calls to external routines that are not written by you.

cwbSV_LogTraceData

Purpose: This function will log the supplied trace data to the currently active trace file. The product and component ID's set in the entry will be written along with the date and time of the when the data was logged.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_LogTraceData(  
    cwbSV_TraceDataHandle traceDataHandle,  
    char *traceData,  
    unsigned long traceDataLength);
```

Parameters:

cwbSV_TraceDataHandle traceDataHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceDataHandle()`.

char * traceData - input

Points to a buffer that contains the trace data you want to log. The buffer can contain binary data because the length parameter is used in determining the amount to trace.

unsigned long traceDataLength - input

Specifies the number of bytes in the trace data buffer to log for this trace entry.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: None

cwbSV_OpenServiceFile

Purpose: Opens the specified service file for READ access (history log, trace file, and so forth) and returns a handle to it.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_OpenServiceFile(  
    char *serviceFileName,  
    cwbSV_ServiceFileHandle *serviceFileHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

char * serviceFileName - input

Points to a buffer that contains the fully-qualified name (for example, c:\path\filename.ext) of the service file to open.

cwbSV_ServiceFileHandle * serviceFileHandle - input/output

Pointer to a cwbSV_ServiceFileHandle where the handle will be returned. This handle should be used in subsequent calls to the service file functions.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_POINTER

NULL passed as handle address.

CWB_FILE_IO_ERROR

File could not be opened.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory to create handle.

Usage: None

cwbSV_ReadNewestRecord

Purpose: Reads the newest record in the service file into the record handle that is provided. Subsequent calls can be made to retrieve the information that is stored in this record (for example, GetProduct(), GetDateStamp(), and so forth). Note: This record is the one with the newest time and date stamp in the file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_ReadNewestRecord(  
    cwbSV_ServiceFileHandle serviceFileHandle,  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the cwbSV_OpenServiceFile function.

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the cwbSV_CreateServiceRecHandle function.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_END_OF_FILE

End of file has been reached.

CWB_FILE_IO_ERROR

Record could not be read.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This read would be used as a "priming-type" read before issuing a series of cwbSV_ReadPrevRecord() calls until the end-of-file indicator is returned.

cwbSV_ReadNextRecord

Purpose: Reads the next record in the service file into the record handle that is provided. Subsequent calls can be made to retrieve the information that is stored in this record (for example, GetProduct(), GetDateStamp(), and so forth).

Syntax:

```
unsigned int CWB_ENTRY cwbSV_ReadNextRecord(  
    cwbSV_ServiceFileHandle serviceFileHandle,  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the cwbSV_OpenServiceFile function.

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the cwbSV_CreateServiceRecHandle function.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_END_OF_FILE

End of file has been reached.

CWB_FILE_IO_ERROR

Record could not be read.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This read would normally be used once the priming read, "ReadOldestRecord()" is performed.

cwbSV_ReadOldestRecord

Purpose: Reads the oldest record in the service file into the record handle that is provided. Subsequent calls can be made to retrieve the information that is stored in this record (for example, `GetProduct()`, `GetDateStamp()`, and so forth). Note: This record is the one with the oldest time and date stamp in the file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_ReadOldestRecord(  
    cwbSV_ServiceFileHandle serviceFileHandle,  
    cwbSV_ServiceRecHandle serviceRecHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the `cwbSV_OpenServiceFile` function.

cwbSV_ServiceRecHandle serviceRecHandle - input

Handle that was returned by a previous call to the `cwbSV_CreateServiceRecHandle` function.

cwbSV_ErrHandle errorHandle - output

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_END_OF_FILE

End of file has been reached.

CWB_FILE_IO_ERROR

Record could not be read.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This read would be used as a "priming-type" read before issuing a series of `cwbSV_ReadNextRecord()` calls until the end-of-file indicator is returned.

cwbSV_ReadPrevRecord

Purpose: Reads the previous record in the service file into the record handle that is provided. Subsequent calls can be made to retrieve the information that is stored in this record (for example, GetProduct(), GetDateStamp(), and so forth).

Syntax:

```
unsigned int CWB_ENTRY cwbSV_ReadPrevRecord(  
    cwbSV_ServiceFileHandle serviceFileHandle,  
    cwbSV_ServiceRechandle serviceRechandle,  
    cwbSV_ErrHandle         errorHandler);
```

Parameters:

cwbSV_ServiceFileHandle serviceFileHandle - input

Handle that was returned by a previous call to the cwbSV_OpenServiceFile function.

V_ServiceRechandle serviceRechandle -input Handle that was returned by a previous call to the cwbSV_CreateServiceRechandle function.

cwbSV_ErrHandle errorHandler - output

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrieved.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_END_OF_FILE

End of file has been reached.

CWB_FILE_IO_ERROR

Record could not be read.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This read would normally be used once the priming read, "ReadNewestRecord()" is performed.

cwbSV_SetMessageClass

Purpose: This function allows setting of the message class (severity) to associate with the message being written to the history log.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetMessageClass(  
    cwbSV_MessageTextHandle messageTextHandle,  
    cwbSV_MessageClass      messageClass);
```

Parameters:

cwbSV_MessageTextHandle messageTextHandle - input

Handle that was returned by a previous call to `cwbSV_CreateMessageTextHandle()`.

cwbSV_MessageClass messageClass - input

One of the following:

`CWBSV_CLASS_INFORMATIONAL`

`CWBSV_CLASS_WARNING`

`CWBSV_CLASS_ERROR`

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Unusable handle passed in on request.

CWBSV_INVALID_MSG_CLASS

Invalid message class passed in.

Usage: This value should be set before calling the corresponding log function, "`cwbSV_LogMessageText()`".

cwbSV_SetMessageComponent

Purpose: This function allows setting of a unique component identifier in the message handle that is provided. Along with setting the product ID (see `cwbSV_SetMessageProduct`), this call should be used to distinguish your message entries from other product's entries in the history log.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetMessageComponent(  
    cwbSV_MessageTextHandle messageTextHandle,  
    char *componentID);
```

Parameters:

cwbSV_MessageTextHandle messageTextHandle - input

Handle that was returned by a previous call to `cwbSV_CreateMessageTextHandle()`.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this message entry. NOTE: A maximum of `CWBSV_MAX_COMP_ID` characters will be logged for the component ID. Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Unusable handle passed in on request.

Usage: This value should be set before calling the corresponding log function, "`cwbSV_LogMessageData()`". The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

cwbSV_SetMessageProduct

Purpose: This function allows setting of a unique product identifier in the message handle that is provided. Along with setting the component ID (see `cwbSV_SetMessageComponent`), this call should be used to distinguish your message entries from other product's entries in the history log.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetMessageProduct(  
    cwbSV_MessageTextHandle messageTextHandle,  
    char *productID);
```

Parameters:

cwbSV_MessageTextHandle messageTextHandle - input

Handle that was returned by a previous call to `cwbSV_CreateMessageTextHandle()`.

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this message entry.

NOTE: A maximum of `CWBSV_MAX_PRODUCT_ID` characters will be logged for the product ID.

Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Unusable handle passed in on request.

Usage: This value should be set before calling the corresponding log function, "`cwbSV_LogMessageData()`". The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

cwbSV_SetAPIComponent

Purpose: This function allows setting of a unique component identifier in trace entry that is provided. Along with setting the product ID (see `cwbSV_SetAPIProduct`), this call should be used to distinguish your trace entries from other product's entries in the trace file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetAPIComponent(  
    cwbSV_TraceAPIHandle traceAPIHandle,  
    char *componentID);
```

Parameters:

cwbSV_TraceAPIHandle traceAPIHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceAPIHandle()`.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this trace entry.

NOTE: A maximum of `CWBSV_MAX_COMP_ID` characters will be logged for the component ID.

Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This value should be set before calling the corresponding log functions, "`cwbSV_LogAPIEntry()`" and "`cwbSV_LogAPIExit()`". The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

cwbSV_SetAPIProduct

Purpose: This function allows setting of a unique product identifier in the trace handle that is provided. Along with setting the component ID (see `cwbSV_SetAPIComponent`), this call should be used to distinguish your trace entries from other product's entries in the trace file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetAPIProduct(  
    cwbSV_TraceAPIHandle traceAPIHandle,  
    char *productID);
```

Parameters:

cwbSV_TraceAPIHandle traceAPIHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceAPIHandle()`.

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this trace entry.

NOTE: A maximum of `CWBSV_MAX_PRODUCT_ID` characters will be logged for the product ID.

Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This value should be set before calling the corresponding log functions, "`cwbSV_LogAPIEntry()`" and "`cwbSV_LogAPIExit()`". The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

cwbSV_SetSPIComponent

Purpose: This function allows setting of a unique component identifier in trace entry that is provided. Along with setting the product ID (see `cwbSV_SetSPIProduct`), this call should be used to distinguish your trace entries from other product's entries in the trace file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetSPIComponent(  
    cwbSV_TraceSPIHandle traceSPIHandle,  
    char *componentID);
```

Parameters:

cwbSV_TraceSPIHandle traceSPIHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceSPIHandle()`.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this trace entry.

NOTE: A maximum of `CWBSV_MAX_COMP_ID` characters will be logged for the component ID.

Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This value should be set before calling the corresponding log functions, "`cwbSV_LogAPIEntry()`" and "`cwbSV_LogAPIExit()`". The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

cwbSV_SetSPIProduct

Purpose: This function allows setting of a unique product identifier in the trace handle that is provided. Along with setting the component ID (see `cwbSV_SetSPIComponent`), this call should be used to distinguish your trace entries from other product's entries in the trace file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetSPIProduct(  
    cwbSV_TraceSPIHandle traceSPIHandle,  
    char *productID);
```

Parameters:

cwbSV_TraceSPIHandle traceSPIHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceSPIHandle()`.

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this trace entry.

NOTE: A maximum of `CWBSV_MAX_PRODUCT_ID` characters will be logged for the product ID.

Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This value should be set before calling the corresponding log functions, "`cwbSV_LogAPIEntry()`" and "`cwbSV_LogAPIExit()`". The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

cwbSV_SetTraceComponent

Purpose: This function allows setting of a unique component identifier in service entry that is provided. Along with setting the product ID (see `cwbSV_SetTraceProduct`), this call should be used to distinguish your trace entries from other product's entries in the trace file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetTraceComponent(  
    cwbSV_TraceDataHandle traceDataHandle,  
    char *componentID);
```

Parameters:

cwbSV_TraceDataHandle traceDataHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceDataHandle()`.

char * componentID - input

Points to a null-terminated string that contains a component identifier to be used on this trace entry.

NOTE: A maximum of `CWBSV_MAX_COMP_ID` characters will be logged for the component ID.

Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This value should be set before calling the corresponding log function, "`cwbSV_LogTraceData()`". The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

cwbSV_SetTraceProduct

Purpose: This function allows setting of a unique product identifier in the trace handle that is provided. Along with setting the component ID (see `cwbSV_SetTraceComponent`), this call should be used to distinguish your trace entries from other product's entries in the trace file.

Syntax:

```
unsigned int CWB_ENTRY cwbSV_SetTraceProduct(  
    cwbSV_TraceDataHandle traceDataHandle,  
    char *productID);
```

Parameters:

cwbSV_TraceDataHandle traceDataHandle - input

Handle that was returned by a previous call to `cwbSV_CreateTraceDataHandle()`.

char * productID - input

Points to a null-terminated string that contains a product identifier to be used on this trace entry.

NOTE: A maximum of `CWBSV_MAX_PRODUCT_ID` characters will be logged for the product ID.

Larger strings will be truncated.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_HANDLE

Handle is not valid.

Usage: This value should be set before calling the corresponding log function, `cwbSV_LogTraceData`. The suggested hierarchy is that you would define a product ID with one or many components that are defined under it.

Example: Using iSeries Access for Windows serviceability APIs

The following example uses the iSeries Access for Windows Serviceability APIs to log a message string to the iSeries Access for Windows History Log:

```
#include <stdio.h>
#include "CWBSV.H"

unsigned int logMessageText(char *msgtxt)
/* Write a message to the active message log. */
{
    cwbSV_MessageTextHandle messageTextHandle;
    unsigned int    rc;

    /* Create a handle to a message text object, so that we may write */
    /* message text to the active message log. */
    if ((rc = cwbSV_CreateMessageTextHandle("ProductID", "ComponentID",
        &messageTextHandle)) != CWB_OK)
        return(rc);

    /* Log the supplied message text to the active message log. */
    rc = cwbSV_LogMessageText(messageTextHandle, msgtxt, strlen(msgtxt));

    /* Delete the message text object identified by the handle provided.*/
    cwbSV_DeleteMessageTextHandle(messageTextHandle);

    return(rc);
}

unsigned int readMessageText(char **bufptr, cwbSV_ErrHandle errorHandler)
/* Read a message from the active message log. */
{
    cwbSV_ServiceFileHandle serviceFileHandle;
    cwbSV_ServiceRecHandle  serviceRecHandle;
    static char buffer[BUFSIZ];
    unsigned int    rc;

    /* Retrieve the fully-qualified path and file name of the active */
    /* message log. */
    if ((rc = cwbSV_GetServiceFileName(CWBSV_HISTORY_LOG, buffer, BUFSIZ,
        NULL)) != CWB_OK)
        return(rc);

    /* Open the active message log for READ access and return a handle */
    /* to it. */
    if ((rc = cwbSV_OpenServiceFile(buffer, &serviceFileHandle, errorHandler))
        != CWB_OK)
        return(rc);

    /* Create a service record object and return a handle to it. */
    if ((rc = cwbSV_CreateServiceRecHandle(&serviceRecHandle)) != CWB_OK) {
        cwbSV_CloseServiceFile(serviceFileHandle, 0);
        return(rc);
    }

    /* Read the newest record in the active message log into the */
    /* record handle provided. */
    if ((rc = cwbSV_ReadNewestRecord(serviceFileHandle, serviceRecHandle,
        errorHandler)) != CWB_OK) {
        cwbSV_DeleteServiceRecHandle(serviceRecHandle);
        cwbSV_CloseServiceFile(serviceFileHandle, 0);
    }
}
```



```

        return(rc);
    }

    /* Retrieve the message text portion of the service record object */
    /* identified by the handle provided. */
    if ((rc = cwbsv_GetMessageText(serviceRecHandle, buffer, BUFSIZ, NULL))
        == CWB_OK || rc == CWB_BUFFER_OVERFLOW) {
        *bufptr = buffer;
        rc = CWB_OK;
    }

    /* Delete the service record object identified by the */
    /* handle provided. */
    cwbsv_DeleteServiceRecHandle(serviceRecHandle);

    /* Close the active message log identified by the handle provided.*/
    cwbsv_CloseServiceFile(serviceFileHandle, errorHandle);

    return(rc);
}

void main(int argc, char *argv[ ])
{
    cwbsv_ErrHandle errorHandle;
    char *msgtxt = NULL, errbuf[BUFSIZ];
    unsigned int rc;

    /* Write a message to the active message log. */
    if (logMessageText("Sample message text") != CWB_OK)
        return;

    /* Create an error message object and return a handle to it. */
    cwbsv_CreateErrHandle(&errorHandle);

    /* Read a message from the active message log. */
    if (readMessageText(&msgtxt, errorHandle) != CWB_OK) {
        if ((rc = cwbsv_GetErrText(errorHandle, errbuf, BUFSIZ, NULL)) ==
            CWB_OK || rc == CWB_BUFFER_OVERFLOW)
            fprintf(stdout, "%s\n", errbuf);
    }
    else if (msgtxt)
        fprintf(stdout, "Message text: \"%s\"\n", msgtxt);

    /* Delete the error message object identified by the */
    /* handle provided. */
    cwbsv_DeleteErrHandle(errorHandle);
}

```

iSeries Access for Windows System Object Access (SOA) APIs

System Object Access enables you to view and manipulate iSeries objects through a graphical user interface. System Object Access application programming interfaces (APIs) for iSeries Access for Windows provide direct access to object attributes. For example, to obtain the number of copies for a given spool file, you can call a series of SOA APIs, and change the value as needed.

System Object Access APIs for iSeries Access for Windows required files:

Interface definition file	Import library	Dynamic Link Library
cwbssoapi.h	cwbsapi.lib	cwbssoapi.dll

Programmer's Toolkit:

The Programmer's Toolkit provides System Object Access documentation, access to the cwboapi.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select **iSeries Operations** → **C/C++ APIs**.

System Object Access APIs for iSeries Access for Windows topics:

- "SOA objects"
- "iSeries object views"
- "Typical use of System Object Access APIs for iSeries Access for Windows"
- **System Object Access APIs for iSeries Access for Windows listing**
- "System Object Access APIs return codes" on page 30

Related topic:

- "iSeries system name formats for ODBC Connection APIs" on page 12

SOA objects

Use System Object Access to view and to manipulate the following iSeries objects:

You can view and manipulate these objects:

- Jobs
- Printers
- Printed output
- Messages
- Spooled files

You only can manipulate these objects:

- Users and groups
- TCP/IP interfaces
- TCP/IP routes
- Ethernet lines
- Token-ring lines
- Hardware resources
- Software resources
- Libraries in QSYS

iSeries object views

Two types of **iSeries object views** are provided with iSeries Access for Windows:

List view:

Displays a customizable graphical list view of the selected iSeries objects. The user can perform a variety of actions on one or more objects.

Properties view:

Displays a detailed graphical view of the attributes of a specific iSeries object. The user can view all attributes if desired, and make changes to those attributes that are changeable.

Typical use of System Object Access APIs for iSeries Access for Windows

Links to three summaries for and examples of System Object Access API usage are provided below. Each example is presented twice; a typical sequence of API calls is shown in summary form, and then an actual C-language sample program is presented. The summary indicates which APIs are required (R) and which are optional (O). Normally, additional code would be required to check for and handle errors on each function call; this has been omitted for illustration purposes.

Typical use of SOA APIs for iSeries Access for Windows summaries and examples:

- "Displaying a customized list of iSeries objects" on page 493
- "Sample program: Displaying a customized list of iSeries objects" on page 493

- “Displaying the Properties view for an iSeries Object” on page 495
- “Sample program: Displaying the Properties view of an object” on page 495
- “Accessing and updating data for iSeries Objects” on page 497
- “Sample program: Accessing and updating data for iSeries objects” on page 498

Displaying a customized list of iSeries objects

A list object for a list of iSeries spool files is created. After setting the desired sort and filter criteria, the list is displayed to the user, with the user interface customized so that certain user actions are disabled. When the user is finished viewing the list, the filter criteria are saved in the application profile and the program exits.

Displaying a customized list of iSeries objects (summary)

(O)	cwBRC_StartSys	Start a iSeries conversation
(R)	CWBSO_CreateListHandle	Create a list of iSeries objects
(O)	CWBSO_SetListProfile	Set name of application
(O)	CWBSO_ReadListProfile	Load application preferences
(O)	CWBSO_SetListFilter	Set list filter criteria
(O)	CWBSO_SetListSortFields	Set list sort criteria
(O)	CWBSO_DisallowListFilter	Do not allow user to change filter criteria
(O)	CWBSO_DisallowListActions	Disallow selected list actions
(O)	CWBSO_SetListTitle	Set title of list
(R)	CWBSO_CreateErrorHandle	Create an error object
(R)	CWBSO_DisplayList	Display the customized list
(O)	CWBSO_DisplayErrMsg	Display error message if error occurred
(O)	CWBSO_WriteListProfile	Save list filter criteria
(R)	CWBSO_DeleteErrorHandle	Delete error object
(R)	CWBSO_DeleteListHandle	Delete list
(O)	cwBRC_StopSys	End iSeries conversation

To view the example:

“Sample program: Displaying a customized list of iSeries objects”

Sample program: Displaying a customized list of iSeries objects

```
#ifndef UNICODE
#define _UNICODE
#endif
#include <windows.h>           // Windows APIs and datatypes
#include "cwbsapi.h"         // System Object Access APIs
#include "cwbrc.h"           // iSeries DPC APIs
#include "cwbn.h"            // iSeries Navigator APIs

#define APP_PROFILE "APPROF" // Application profile name

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR lpszCmdLine, int nCmdShow)
{
    MSG          msg;           // Message structure
    HWND         hWnd;         // Window handle
    cwBRC_SysHandle hSystem;    // System handle
    CWBSO_LIST_HANDLE hList = CWBSO_NULL_HANDLE; // List handle
    CWBSO_ERR_HANDLE hError = CWBSO_NULL_HANDLE; // Error handle
    cwCO_SysHandle hSystemHandle; // System object handle
    unsigned int rc;           // System Object Access return codes
```

```

unsigned short    sortIDs[] = { CWBSO_SFL_SORT_UserData,
                               CWBSO_SFL_SORT_Priority };
// Array of sort IDs
unsigned short    actionIDs[] = { CWBSO_ACTN_PROPERTIES };
// Array of action IDs

//*****
// Start a conversation with iSeries server SYSNAME. Specify
// application name APPNAME.
//*****
cwbUN_GetSystemHandle((char *)"SYSNAME", (char *)"APPNAME", &hSystemHandle);

cwbRC_StartSysEx(hSystemHandle, &hSystem);

//*****
// Create a list of spooled files. Set desired sort/filter criteria.

// Create a list of spooled files on system SYSNAME
CWBSO_CreateListHandleEx(hSystemHandle,
                        CWBSO_LIST_SFL,
                        &hList);

// Identify the name of the application profile
CWBSO_SetListProfile(hList, APP_PROFILE);

// Create an error handle
CWBSO_CreateErrorHandle(&hError);

// Load previous filter criteria
CWBSO_ReadListProfile(hList, hError);

// Only show spooled files on printer P3812 for user TLK
CWBSO_SetListFilter(hList, CWBSO_SFLF_DeviceFilter, "P3812");
CWBSO_SetListFilter(hList, CWBSO_SFLF_UserFilter, "TLK");

// Sort by 'user specified data', then by 'output priority'
CWBSO_SetListSortFields(hList, sortIDs, sizeof(sortIDs) / sizeof(short));

//*****
// Customize the UI by disabling selected UI functions. Set the list title.
//*****

// Do not allow users to change list filter
CWBSO_DisallowListFilter(hList);

// Do not allow the 'properties' action to be selected
CWBSO_DisallowListActions(hList, actionIDs, sizeof(actionIDs) / sizeof(short));

// Set the string that will appear in the list title bar
CWBSO_SetListTitle(hList, "Application Title");

//*****
// Display the list.
//*****

// Display the customized list of spooled files
rc = CWBSO_DisplayList(hList, hInstance, nCmdShow, &hWnd, hError);

// If an error occurred, display a message box
if (rc == CWBSO_ERROR_OCCURRED)
    CWBSO_DisplayErrMsg(hError);
else
{
    // Dispatch messages for the list window
    while(GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    // List window has been closed - save filter criteria in application profile
    CWBSO_WriteListProfile(hList, hError);
}

//*****

```

```

// Processing complete - clean up and exit.
//*****

// Clean up handles
CWBSO_DeleteErrorHandle(hError);
CWBSO_DeleteListHandle(hList);

// End the conversation started by EHNDP_StartSys
cwbRC_StopSys(hSystem);

//*****
// Return from WinMain.
//*****

return rc;
}

```

Displaying the Properties view for an iSeries Object

A list object for a list of iSeries spool files is created. After setting the desired filter criteria, the list is opened, and a handle to the first object in the list is obtained. A properties view that shows the attributes for this object is displayed to the user.

Displaying the properties view for an object (Summary)

(O)	cwbRC_StartSys	Start a conversation with an iSeries server
(R)	CWBSO_CreateListHandle	Create a list of iSeries objects
(O)	CWBSO_SetListFilter	Set list filter criteria
(R)	CWBSO_CreateErrorHandle	Create an error object
(R)	CWBSO_OpenList	Open the list (builds a list on the iSeries server)
(O)	CWBSO_DisplayErrMsg	Display error message if error occurred
(O)	CWBSO_GetListSize	Get number of objects in the list
(R)	CWBSO_GetObjHandle	Get an object from the list
(R)	CWBSO_DisplayObjAttr	Display the properties view for the object
(R)	CWBSO_DeleteObjHandle	Delete the object
(O)	CWBSO_CloseList	Close the list
(R)	CWBSO_DeleteErrorHandle	Delete error object
(R)	CWBSO_DeleteListHandle	Delete list
(O)	cwbRC_StopSys	End iSeries conversation

To view the example:

“Sample program: Displaying the Properties view of an object”

Sample program: Displaying the Properties view of an object

```

#ifndef UNICODE
#define _UNICODE
#endif
#include <windows.h> // Windows APIs and datatypes
#include "cwboapi.h" // System Object Access APIs
#include "cwbrc.h" // iSeries DPC APIs
#include "cwbn.h" // iSeries Navigator APIs

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpszCmdLine, int nCmdShow)
{
MSG msg; // Message structure
HWND hWnd; // Window handle

```

```

cwbRC_SysHandle  hSystem;           // System handle
CWBSO_LIST_HANDLE hList = CWBSO_NULL_HANDLE; // List handle
CWBSO_ERR_HANDLE hError = CWBSO_NULL_HANDLE; // Error handle
CWBSO_OBJ_HANDLE hObject = CWBSO_NULL_HANDLE; // Object handle
cwbCO_SysHandle  hSystemHandle;     // System object handle
unsigned long    listSize = 0;      // List size
unsigned short   listStatus = 0;    // List status
unsigned int     rc;                // System Object Access return codes

//*****
// Start a conversation with iSeries server SYSNAME. Specify
// application name APPNAME.
//*****

cwbUN_GetSystemHandle((char *)"SYSNAME", (char *)"APPNAME", &hSystemHandle);

cwbRC_StartSysEx(hSystemHandle, &hSystem);

//*****
// Create a list of spooled files. Set desired filter criteria.
//*****

// Create a list of spooled files on system SYSNAME
CWBSO_CreateListHandleEx(hSystemHandle,
                        CWBSO_LIST_SFL,
                        &hList);

// Only include spooled files on printer P3812 for user TLK
CWBSO_SetListFilter(hList, CWBSO_SFLF_DeviceFilter, "P3812");
CWBSO_SetListFilter(hList, CWBSO_SFLF_UserFilter, "TLK");

//*****
// Open the list.
//*****

// Create an error handle
CWBSO_CreateErrorHandle(&hError);

// Open the list of spooled files
rc = CWBSO_OpenList(hList, hError);
// If an error occurred, display a message box
if (rc == CWBSO_ERROR_OCCURRED)
    CWBSO_DisplayErrMsg(hError);
else
{
    //*****
    // Display the properties of the first object in the list
    //*****

    // Get the number of objects in the list
    CWBSO_GetListSize(hList, &listSize, &listStatus, hError);

    if (listSize > 0)
    {
        // Get the first object in the list
        CWBSO_GetObjHandle(hList, 0, &hObject, hError);

        // Display the properties window for this object
        CWBSO_DisplayObjAttr(hObject, hInstance, nCmdShow, &hWnd, hError);

        // Dispatch messages for the properties window
        while(GetMessage(&msg, NULL, 0, 0))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }

        // Properties window has been closed - delete object handle
        CWBSO_DeleteObjHandle(hObject);
    }
}

//*****
// Processing complete - clean up and exit.
//*****

```

```

// Close the list
CWBSO_CloseList(hList, hError);

// Clean up handles
CWBSO_DeleteErrorHandle(hError);
CWBSO_DeleteListHandle(hList);

// End the conversation started by EHNDP_StartSys
cwbRC_StopSys(hSystem);

//*****
// Return from WinMain.
//*****

return rc;
}

```

Accessing and updating data for iSeries Objects

In “Sample program: Accessing and updating data for iSeries objects” on page 498, all spooled files for device P3812 that have 10 or more pages have their output priority changed to 9 so that they will not print before smaller files.

A list object for a list of iSeries spool files is created. After setting the desired filter criteria, the list is opened. A parameter object is created which will be used to change the output priority for each spooled file in the list. After storing the desired output priority value of “9” in the parameter object, a loop is entered. Each object in the list is examined in turn, and if a spooled file is found to have more than 10 pages then its output priority is changed.

Accessing and updating data for iSeries objects (Summary)

(R)	CWBSO_CreateListHandle	Create a list of iSeries objects
(O)	CWBSO_SetListFilter	Set list filter criteria
(R)	CWBSO_CreateErrorHandle	Create an error object
(R)	CWBSO_OpenList	Open the list (automatically starts a conversation with the iSeries server)
(O)	CWBSO_DisplayErrMsg	Display error message if error occurred
(R)	CWBSO_CreateParmObjHandle	Create a parameter object
(R)	CWBSO_SetParameter	Set new value for object attribute or attributes
(R)	CWBSO_WaitForObj	Wait until first object is available
	. . . Loop through all objects	
	.	
.	(R) CWBSO_GetObjHandle	Get an object from the list
.	(R) CWBSO_GetObjAttr	Read data for a particular attribute
.	(R) CWBSO_SetObjAttr	Update an attribute on the iSeries server
.	(R) CWBSO_DeleteObjHandle	Clean up object handle
.	(R) CWBSO_WaitForObj	Wait for next object in list
.	
(R)	CWBSO_DeleteParmObjHandle	Delete the parameter object
(O)	CWBSO_CloseList	Close the list

- (R) CWBSO_DeleteErrorHandle Delete error object
- (R) CWBSO_DeleteListHandle Delete list (automatically ends the iSeries conversation)

To view the example:

“Sample program: Accessing and updating data for iSeries objects”

Sample program: Accessing and updating data for iSeries objects

```
#include <windows.h>                    // Windows APIs and datatypes
#include <stdlib.h>                    // For atoi
#include "cwbssoapi.h"                // System Object Access APIs

int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                  LPSTR lpszCmdLine, int nCmdShow)
{
CWBSO_LIST_HANDLE hList = CWBSO_NULL_HANDLE;    // List handle
CWBSO_ERR_HANDLE hError = CWBSO_NULL_HANDLE;   // Error handle
CWBSO_PARMOBJ_HANDLE hParmObject = CWBSO_NULL_HANDLE; // Parm object
CWBSO_OBJ_HANDLE hObject = CWBSO_NULL_HANDLE;   // Object handle
unsigned int rc, setRC;                // System Object Access return codes
unsigned long bytesNeeded = 0;         // Bytes needed
unsigned short errorIndex = 0;         // Error index (SetObjAttr)
char szString[100]; // Buffer for formatting
int totalPages = 0;                    // Total pages
int i = 0;                            // Loop counter
int nNbrChanged = 0;                 // Count of changed objects

MessageBox(GetFocus(), "Start of Processing", "PRIORITY", MB_OK);

//*****
// Create a list of spooled files. Set desired filter criteria.
//*****

// Create a list of spooled files on system SYSNAME
CWBSO_CreateListHandle("SYSNAME",
                      "APPNAME",
                      CWBSO_LIST_SFL,
                      &hList);

// Only include spooled files for device P3812
CWBSO_SetListFilter(hList, CWBSO_SFLF_DeviceFilter, "P3812");

//*****
// Open the list.
//*****

// Create an error handle
CWBSO_CreateErrorHandle(&hError);

// Open the list of spooled files
rc = CWBSO_OpenList(hList, hError);

// If an error occurred, display a message box
if (rc == CWBSO_ERROR_OCCURRED)
    CWBSO_DisplayErrMsg(hError);
else
{
    //*****
    // Set up to change output priority for all objects in the list.
    //*****

    // Create a parameter object to hold the attribute changes
    CWBSO_CreateParmObjHandle(&hParmObject);

    // Set the parameter to change the output priority to '9'
```



```

CWBSO_SetParameter(hParmObject,
                  CWBSO_SFL_OutputPriority,
                  "9",
                  hError);

//*****
// Loop through the list, changing the output priority for any
// files that have more than 10 total pages. Loop will
// terminate when CWBSO_WaitForObj
// returns CWBSO_BAD_LIST_POSITION, indicating that there
// are no more objects in the list.
//*****

// Wait for first object in the list
rc = CWBSO_WaitForObj(hList, i, hError);

// Loop through entire list
while (rc == CWBSO_NO_ERROR)
{
    // Get the list object at index i
    CWBSO_GetObjHandle(hList, i, &hObject, hError);

    // Get the total pages attribute for this spooled file
    CWBSO_GetObjAttr(hObject,
                    CWBSO_SFL_TotalPages,
                    szString,
                    sizeof(szString),
                    &bytesNeeded,
                    hError);

    totalPages = atoi(szString);

    // Update the output priority if necessary
    if (totalPages > 10)
    {
        // Change the spool file's output priority to '9'
        setRC = CWBSO_SetObjAttr(hObject, hParmObject, &errorIndex, hError);
        if (setRC == CWBSO_NO_ERROR)
            nNbrChanged++;
    }

    // Delete the object handle
    CWBSO_DeleteObjHandle(hObject);

    // Increment list item counter
    i++;

    // Wait for next list object
    rc = CWBSO_WaitForObj(hList, i, hError);
} /* end while */

// Parameter object no longer needed
CWBSO_DeleteParmObjHandle(hParmObject);

} /* end if */

// Display the number of spooled files that had priority changed
wprintf(szString, "Number of spool files changed: %d", nNbrChanged);
MessageBox(GetFocus(), szString, "PRIORITY", MB_OK);

//*****
// Processing complete - clean up and exit.
//*****

```

```

// Close the list
CWBSO_CloseList(hList,hError);

// Clean up handles
CWBSO_DeleteErrorHandle(hError);
CWBSO_DeleteListHandle(hList);

//*****
// Return from WinMain.
//*****

return 0;
}

```

iSeries Access for Windows System Object Access programming considerations

See the following topics for important SOA programming considerations:

- “About System Object Access errors”
- “System Object Access application profiles”
- “Managing iSeries communications sessions for application programs”

About System Object Access errors

All System Object Access APIs use return codes to report error conditions. Check for errors on each function call. In addition, certain APIs incorporate a handle to an “error object” in their interface. The error object is used to provide additional information for errors which occurred during the processing of a request. Often these errors are encountered while interacting with the iSeries server, in which case the error object will contain the error message text.

If a function call returns CWBSO_ERROR_OCCURRED then the error object will have been filled in with information that describe the error. CWBSO_GetErrMsgText may be used to retrieve the error message text. The message will have been translated into the language that is specified for the user’s execution environment. Alternatively, the error message may be displayed to the user directly by calling CWBSO_DisplayErrMsg.

For internal processing errors, error objects automatically log an entry in the System Object Access log file soa.log, in the iSeries Access for Windows install directory. This file is English only and is intended for use by IBM personnel for problem analysis.

Related topic:

“System Object Access APIs return codes” on page 30

System Object Access application profiles

By default, user-specified list filter criteria are not saved to disk. System Object Access provides APIs for:

- Requesting the use of an application-specific registry key for loading the filter data from the registry into a given list object
- Saving the data for a particular list object in the registry

The data is saved by iSeries system name, and within system name by object type. To read or write profile data, a system name must be specified on the CWBSO_CreateListHandle call for the list object.

Managing iSeries communications sessions for application programs

System Object Access APIs for iSeries Access for Windows communicate with the iSeries server through the use of one or more client/server conversations. Because it often takes several seconds to establish a conversation, your application may experience delays when a list first is opened. This topic explains how to control and manage the initiation of conversations so that the performance impact on application programs is minimized.

The default behavior of System Object Access may be summarized as follows:

- If no conversation has been established with the iSeries system object that is identified on the **CWBSO_CreateListHandleEx** API, a conversation automatically will be started when the list is opened or displayed. If iSeries Access for Windows has not yet established a connection to the specified system, a dialog box will appear prompting the user for the appropriate UserID and password.
- If another instance of the application program starts, the above process repeats itself. No conversation sharing occurs between application programs that run in different processes (that is, with different instance handles).
- When the application program deletes the last System Object Access list, the conversation with the iSeries server is automatically ended (Note that **CWBSO_CloseList** does not end the conversation with the iSeries server).

A System Object Access conversation may be started using the **cwbRC_StartSysEx** API. This API accepts an iSeries system object as a parameter, and returns a system handle. Save this handle for later use on the **cwbRC_StopSys** API, when the application is terminating and it is time to end the conversation with the iSeries server.

When the **cwbRC_StartSysEx** API is called, the application is blocked until the conversation is established. Therefore, it is good practice to inform the user that a connection is about to be attempted immediately before the call. On return, the conversation will have been initiated, and System Object Access list processing will use this conversation instead of starting a new one.

When **cwbRC_StartSysEx** is used in this way, the last list to be deleted will not end the conversation. You must call **cwbRC_StopSys** explicitly before you exit the application.

System Object Access APIs for iSeries Access for Windows listing

The following System Object Access APIs for iSeries Access for Windows are listed alphabetically:

System Object Access APIs for iSeries Access for Windows	
CWBSO_CloseList	CWBSO_GetErrMsgText
CWBSO_CopyObjHandle	CWBSO_GetListSize
CWBSO_CreateErrorHandle	CWBSO_GetObjAttr
CWBSO_CreateListHandle	CWBSO_GetObjHandle
CWBSO_CreateListHandleEx	CWBSO_OpenList
CWBSO_CreateObjHandle	CWBSO_ReadListProfile
CWBSO_CreateParmObjHandle	CWBSO_RefreshObj
CWBSO_DeleteErrorHandle	CWBSO_ResetParmObj
CWBSO_DeleteListHandle	CWBSO_SetListFilter
CWBSO_DeleteObjHandle	CWBSO_SetListProfile
CWBSO_DeleteParmObjHandle	CWBSO_SetListSortFields
CWBSO_DisallowListActions	CWBSO_SetListTitle
CWBSO_DisallowListFilter	CWBSO_SetObjAttr
CWBSO_DisplayErrMsg	CWBSO_SetParameter
CWBSO_DisplayList	CWBSO_WaitForObj
CWBSO_DisplayObjAttr	CWBSO_WriteListProfile

See "SOA attribute special values" on page 541 for related information.

SOA enablers:

System Object Access also includes enablers (APIs), which applications can use to access data in iSeries objects or to request graphical lists and attribute views of the object data. The APIs for manipulating lists of objects must be called in the correct order. The basic flow is as follows:

```

CreateErrorHandle -- Creates a handle to an "error" object
                    to be passed to other APIs
CreateListHandle -- Instantiates a list object on the client
OpenList -- Builds list on iSeries server associated with client
            list
(Manipulate the list and its objects using various generic

```

and subclass APIs)
CloseList -- Closes list and release resource on iSeries server
DeleteListHandle -- Destroys list object on the client

The “CWBSO_CreateListHandle” on page 506 API must be called to create a list before any other list APIs are called. The **CWBSO_CreateListHandle** API returns a list handle to the caller. The list handle must be passed as input to all other list APIs.

After the list is allocated, the “CWBSO_SetListFilter” on page 533 API can be called to change the filter criteria for the list. **CWBSO_SetListFilter** is optional; if it is not called, the list will be built with the default filter criteria. Similarly, the “CWBSO_SetListSortFields” on page 535 API can be called to define the attributes on which the list will be sorted. If it is not called the list will not be sorted.

The “CWBSO_OpenList” on page 529 API must be called to build the list of objects. This will result in a request to be sent to the iSeries server. The list will be built on the iSeries server, and some or all of the objects (records) in the list will be buffered down to the list on the client. Although all objects in the list are not necessarily cached on the client, the APIs will behave as if they are. Once the **CWBSO_OpenList** API is called successfully, the following APIs can be called:

“CWBSO_GetObjHandle” on page 527

Retrieves a handle to a specific object in the list. The object handle can then be used to manipulate the specific object.

“CWBSO_DeleteObjHandle” on page 514

Releases the handle returned by **CWBSO_GetObjHandle**.

“CWBSO_DisplayList” on page 519

Displays the spreadsheet view of the list.

“CWBSO_GetListSize” on page 524

Retrieves the number of objects in the list.

“CWBSO_CloseList” on page 503

Closes the list on the iSeries server and destroy all client objects in the list. All object handles returned by **CWBSO_GetListObject** no longer are valid after the list is closed. After the list is closed, the APIs in this list cannot be called until the “CWBSO_OpenList” on page 529 API is called again. The “CWBSO_DeleteListHandle” on page 513 API should be called to destroy the list object.

CWBSO_CloseList

Purpose: Closes the list of objects and frees up resources allocated on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_CloseList(  
    CWBSO_LIST_HANDLE listHandle,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error that was returned by a previous call to CWBSO_CreateErrorHandle. When the value that is returned by this API is CWBSO_ERROR_OCCURRED, the error handle may be used to retrieve the error message text or display the error to the user.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_ERROR_OCCURRED

An error occurred. Use the error handle for more information.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API. The list must currently be open. The list is opened by calling CWBSO_OpenList. This API will not end the conversation with the iSeries server. For the conversation to be ended the list must be deleted using CWBSO_DeleteListHandle.

CWBSO_CopyObjHandle

Purpose: Creates a new instance of an object and returns a handle to the new instance. This does not create a new object on the iSeries server. It merely creates an additional instance of an iSeries object on the client. Object handles that are returned by CWBSO_GetObjHandle are always destroyed when the list that contains the object is closed. This API allows the creation of an instance of the object that will persist after the list is closed. The object instance that was created by this API is kept in sync with the object in the list. In other words, if one of the objects is changed, the changes will be apparent in the other object.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_CopyObjHandle(  
    CWBSO_OBJ_HANDLE objectHandle,  
    CWBSO_OBJ_HANDLE far* lpNewObjectHandle);
```

Parameters:

CWBSO_OBJ_HANDLE objectHandle - input

A handle to an object that was returned by a previous call to CWBSO_GetObjHandle or CWBSO_CopyObjHandle.

CWBSO_OBJ_HANDLE far* lpNewObjectHandle - output

A long pointer to a handle which will be set to a new handle for the same iSeries object. This handle may be used with any other API that accepts an object handle with the exception that some APIs only operate on specific types of objects.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_BAD_OBJ_HANDLE

The object handle that is specified is not valid.

Usage: CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be called prior to calling this API. The object handle that is returned by CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be passed as input to this API. When the object is no longer needed, the calling program is responsible for doing the following:

- Call CWBSO_DeleteObjHandle to free up resources that are allocated on the client.

CWBSO_CreateErrorHandle

Purpose: Creates an error handle. An error handle is used to contain error messages that are returned from other APIs. The error handle may be used to display the error in a dialog or retrieve the associated error message text.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_CreateErrorHandle(  
    CWBSO_ERR_HANDLE far* lpErrorHandle);
```

Parameters:

CWBSO_ERR_HANDLE far* lpErrorHandle - output

A long pointer to a handle which will be set to the handle for an error.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

Usage: When the error handle is no longer needed, the calling program is responsible for doing the following:

- Call CWBSO_DeleteErrorHandle to free up resources that are allocated on the client.

CWBSO_CreateListHandle

Purpose: Creates a new list and returns a handle to the list.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_CreateListHandle(  
    char far* lpszSystemName,  
    char far* lpszApplicationName,  
    CWBSO_LISTTYPE type,  
    CWBSO_LIST_HANDLE far* lpListHandle);
```

Parameters:

char far* lpszSystemName - input

The name of the iSeries system on which the list will be built. The name that is specified must be a configured iSeries server. If the client is not currently connected to the iSeries server, a connection will be established when the list is opened. If NULL is specified for the system name, the current iSeries Access default system will be used.

char far* lpszApplicationName - input

A character string that identifies the application that will be interacting with the list. The maximum length of this string is 10 characters, excluding the NULL terminator.

CWBSO_LISTTYPE type - input

The type of list to be built. Specify one of the following:

CWBSO_LIST_JOB

List of jobs.

CWBSO_LIST_SJOB

List of server jobs.

CWBSO_LIST_SJOB

List of server jobs.

CWBSO_LIST_MSG

List of messages.

CWBSO_LIST_PRT

List of printers.

CWBSO_LIST_SFL

List of spooled files.

CWBSO_LIST_IFC

List interfaces.

CWBSO_LIST_ELN

List Ethernet lines.

CWBSO_LIST_TLN

List token-ring lines.

CWBSO_LIST_HWL

List hardware resources.

CWBSO_LIST_SW

List software products.

CWBSO_LIST_RTE

List TCP/IP route.

CWBSO_LIST_PRF

List user profiles.

CWBSO_LIST_SMP

List libraries in QSYS.

CWBSO_LIST_HANDLE far* IpListHandle - output

A long pointer to a handle that will be set to the handle for the newly created list. This handle may be used with any other API that accepts a list handle.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LISTTYPE

The value that is specified for type of list is not valid.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_BAD_SYSTEM_NAME

The system name that is specified is not a valid iSeries system name.

Usage: When the list is no longer needed, the calling program is responsible for doing the following:

- Call CWBSO_DeleteListHandle to free up resources that are allocated on the client.

CWBSO_CreateListHandleEx

Purpose: Creates a new list and returns a handle to the list.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_CreateListHandleEx(  
    cwbCO_SysHandle systemObjectHandle,  
    CWBSO_LISTTYPE type,  
    CWBSO_LIST_HANDLE far* lpListHandle);
```

Parameters:

cwbCO_SysHandle systemObjectHandle - input

A handle to the system object that represents the iSeries system on which the list will be built. The handle specified must be for a configured iSeries server.

CWBSO_LISTTYPE

The type of list to be built. Specify one of the following:

CWBSO_LIST_JOB

List of jobs.

CWBSO_LIST_SJOB

List of server jobs.

CWBSO_LIST_SJOB

List of server jobs.

CWBSO_LIST_MSG

List of messages.

CWBSO_LIST_PRT

List of printers.

CWBSO_LIST_SFL

List of spooled files.

CWBSO_LIST_IFC

List interfaces.

CWBSO_LIST_ELN

List Ethernet lines.

CWBSO_LIST_TLN

List token-ring lines.

CWBSO_LIST_HWL

List hardware resources.

CWBSO_LIST_SW

List software products.

CWBSO_LIST_RTE

List TCP/IP route.

CWBSO_LIST_PRF

List user profiles.

CWBSO_LIST_SMP

List libraries in QSYS.

CWBSO_LIST_HANDLE far* lpListHandle - output

A long pointer to a handle that will be set to the handle for the newly created list. This handle may be used with any other API that accepts a list handle.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LISTTYPE

The value that is specified for type of list is not valid.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_BAD_SYSTEM_NAME

The system name that is specified is not a valid iSeries system name.

Usage: When the list is no longer needed, the calling program is responsible for doing the following:

- Call CWBSO_DeleteListHandle to free up resources that are allocated on the client.

CWBSO_CreateObjHandle

Purpose: Creates a new object handle and returns a handle to the object. Use this API to access remote object that do not conform to the list format.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_CreateObjHandle(  
    char far* lpszSystemName,  
    char far* lpszApplicationName,  
    CWBSO_OBJTYPE type,  
    CWBSO_OBJ_HANDLE far* lpObjHandle);
```

Parameters:

char far* lpszSystemName - input

The name of the iSeries system on which the object will be built. The name that is specified must be a configured iSeries server. If the client is not currently connected to the iSeries, a connection will be established when the list is opened. If NULL is specified for the system name, the current iSeries default system will be used.

char far* lpszApplicationName - input

A character string that identifies the application that will be interacting with the list. The maximum length of this string is 10 characters, excluding the NULL terminator.

CWBSO_OBJTYPE type - input

The type of object to be built. Specify the following:

- CWBSO_OBJ_TCIPATTR - TCP/IP attributes

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_BAD_SYSTEM_NAME

The system name that is specified is not a valid iSeries system name.

Usage: When the list is no longer needed, the calling program is responsible for doing the following:

- Call CWBSO_DeleteObjHandle to free up resources that are allocated on the client.

CWBSO_CreateParmObjHandle

Purpose: Creates a parameter object and returns a handle to the object. A parameter object contains a set of parameter IDs and values which may be passed as input to other APIs.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_CreateParmObjHandle(  
    CWBSO_PARMOBJ_HANDLE far* lpParmObjHandle);
```

Parameters:

CWBSO_PARMOBJ_HANDLE far* lpParmObjHandle - output

A long pointer to a handle which will be set to the handle for the new parameter object.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

Usage: When the parameter object is no longer needed, the calling program is responsible for doing the following:

- Call CWBSO_DeleteParmObjHandle to free up resources that are allocated on the client.

CWBSO_DeleteErrorHandle

Purpose: Deletes an error handle and frees up resources allocated on the client.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DeleteErrorHandle(  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_ERR_HANDLE errorHandle - input

An error handle that is returned by a previous call to CWBSO_CreateErrorHandle.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

Usage: CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API.

CWBSO_DeleteListHandle

Purpose: Deletes the list of objects and frees up resources allocated on the client.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DeleteListHandle(  
    CWBSO_LIST_HANDLE listHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that is returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API.

CWBSO_DeleteObjHandle

Purpose: Deletes an object handle returned from a previous call to CWBSO_GetObjHandle or CWBSO_CopyObjHandle.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DeleteObjHandle(  
    CWBSO_OBJ_HANDLE objectHandle);
```

Parameters:

CWBSO_OBJ_HANDLE objectHandle - input

A handle to an object that is returned by a previous call to CWBSO_GetObjHandle or CWBSO_CopyObjHandle.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_OBJ_HANDLE

The object handle that is specified is not valid.

Usage: CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be called prior to calling this API. The object handle that is returned by CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be passed as input to this API.

CWBSO_DeleteParmObjHandle

Purpose: Deletes a parameter object handle and frees up resources allocated on the client.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DeleteParmObjHandle(  
    CWBSO_PARMOBJ_HANDLE parmObjHandle);
```

Parameters:

CWBSO_PARMOBJ_HANDLE parmObjHandle - input

A handle to a parameter object that is returned by a previous call to CWBSO_CreateParmObjHandle.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_PARMOBJ_HANDLE

The parameter object handle that is specified is not valid.

Usage: CWBSO_CreateParmObjHandle must be called prior to calling this API. The parameter object handle that is returned by CWBSO_CreateParmObjHandle must be passed as input to this API.

CWBSO_DisallowListActions

Purpose: Sets actions the user is not allowed to perform on objects in a list. This affects the actions available when the list is displayed by calling CWBSO_DisplayList. Disallowed actions do not appear in the menu bar, tool bar, or object pop-up menus. This API can only be called once for a list, and it must be called prior to displaying the list.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DisallowListActions(  
    CWBSO_LIST_HANDLE listHandle,  
    unsigned short far* lpusActionIDs,  
    unsigned short usCount);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that is returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

unsigned short far* lpusActionIDs - input

A long pointer to an array of action identifier values. These values identify which actions the user will not be allowed to perform. The valid values for this parameter depend on the type of objects in the list. See the appropriate header files for the valid values:

- cwbsobjob.h
- cwbsomsg.h
- cwbsopr.h
- cwbsosfl.h

unsigned short usCount - input

The number of action identifier values specified.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ACTION_ID

An action ID specified is not valid for the type of list.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_NOT_ALLOWED_NOW

The action that was requested is not allowed at this time.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API.

CWBSO_DisallowListFilter

Purpose: Sets the list to disallow the user from changing the filter values for the list. This disables the INCLUDE choice from the VIEW pull-down menu when the list is displayed. The list is displayed by calling CWBSO_DisplayList. This API is only meaningful for lists which are displayed by using the CWBSO_DisplayList API. This API can only be called once for a list, and it must be called prior to displaying the list.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DisallowListFilter(  
    CWBSO_LIST_HANDLE listHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that is returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API.

CWBSO_DisplayErrMsg

Purpose: Displays an error message in a dialog box. This API should only be called when CWBSO_ERROR_OCCURRED is the return value from a call to another API. In this case, there is an error message that is associated with the error handle.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DisplayErrMsg(  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_NO_ERROR_MESSAGE

The error handle that is specified contains no error message.

CWBSO_DISP_MSG_FAILED

The request to display the message failed.

Usage: CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API.

CWBSO_DisplayList

Purpose: Displays the list in a window. From this window, the user is allowed to perform actions on the objects in the list.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DisplayList(  
    CWBSO_LIST_HANDLE listHandle,  
    HINSTANCE hInstance,  
    int nCmdShow,  
    HWND far* lphWnd ,  
    CWBSO_ERR_HANDLE errorHandler);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

HINSTANCE hInstance - input

The program instance passed to the calling program's WinMain procedure.

int nCmdShow - input

The show window parameter passed to the calling program's WinMain procedure. Alternatively, any of the constants defined for the Windows API ShowWindow() may be used.

HWND far* lphWnd - output

A long pointer to a window handle. This will be set to the handle of the window in which the list is displayed.

CWBSO_ERR_HANDLE errorHandler - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message text or display the error to the user.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_DISPLAY_FAILED

The window could not be created.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_CreateErrorHandler must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandler must be passed as input to this API. It is not necessary to call CWBSO_OpenList or CWBSO_CloseList when using this API. CWBSO_DisplayList handles both the opening and closing of the list. Your program must have a message loop to receive the Windows messages that will be sent during the use of the system object list.

This API only applies to the following list types: Jobs, Messages, Printers, Printer Output, and Spooled Files.

CWBSO_DisplayObjAttr

Purpose: Displays the attributes window for an object. From this window, the user is allowed to view the object attributes and change attributes that are changeable.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_DisplayObjAttr(  
    CWBSO_OBJ_HANDLE objectHandle,  
    HINSTANCE hInstance,  
    int nCmdShow,  
    HWND far* lphWnd ,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_OBJ_HANDLE objectHandle - input

A handle to an object that was returned by a previous call to CWBSO_GetObjHandle or CWBSO_CopyObjHandle.

HINSTANCE hInstance - input

The program instance passed to the calling program's WinMain procedure.

int nCmdShow - input

The show window parameter passed to the calling program's WinMain procedure. Alternatively, any of the constants defined for the Windows API ShowWindow() may be used.

HWND far* lphWnd - output

A long pointer to a window handle. This will be set to the handle of the window in which the object attributes are displayed.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_OBJ_HANDLE

The object handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_DISPLAY_FAILED

The window could not be created.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be called prior to calling this API. The object handle that is returned by CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API. Your program must have a message loop to receive the Windows messages that will be sent during the use of the system object attributes window.

This API only applies to the following list types: Jobs, Messages, Printers, Printer Output, and Spooled Files.

CWBSO_GetErrMsgText

Purpose: Retrieves the message text from an error handle. This API should only be called when CWBSO_ERROR_OCCURRED is the return value from a call to another API. In this case there is an error message associated with the error handle.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_GetErrMsgText(  
    CWBSO_ERR_HANDLE errorHandle ,  
    char far* lpszMsgBuffer ,  
    unsigned long ulBufferLength,  
    unsigned long far* lpulBytesNeeded);
```

Parameters:

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

char far* lpszMsgBuffer - output

A long pointer to the output buffer where the message text will be placed. The message text that is returned by this API will be translated text. The output buffer is not changed when the return code is not set to CWBSO_NO_ERROR.

unsigned long ulBufferLength - input

The size, in bytes, of the output buffer argument.

unsigned long far* lpulBytesNeeded - output

A long pointer to an unsigned long that will be set to the number of bytes needed to place the entire message text in the output buffer. When this value is less than or equal to the size of output buffer that is specified, the entire message text is placed in the output buffer. When this value is greater than the size of output buffer that is specified, the output buffer contains a null string. The output buffer is not changed beyond the bytes that are needed for the message text. This value is set to zero when the return code is not set to CWBSO_NO_ERROR.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_NO_ERROR_MESSAGE

The error handle that is specified contains no error message.

CWBSO_GET_MSG_FAILED

The error message text could not be retrieved.

Usage: CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API. For errors which occurred on the iSeries server, the message text will be in the language that is specified for the user's execution environment. All other message text will be in the language that is specified in the Windows Control Panel on the user's personal computer.

CWBSO_GetListSize

Purpose: Retrieves the number of objects in a list.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_GetListSize(  
    CWBSO_LIST_HANDLE listHandle,  
    unsigned long far* lpulSize,  
    unsigned short far* lpusStatus,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

unsigned long far* lpulSize - output

A long pointer to an unsigned long that will be set to the number of entries currently in the list. If the list status indicates that the list is complete, this value represents the total number of objects for the list. If the list status indicates that the list is not completely built, this value represents the number of objects currently available from the host and a subsequent call to this API may indicate that more entries are available.

unsigned short far* lpusStatus - output

A long pointer to an unsigned short that will be set to indicate whether the list is completely built. The value will be set to 0 if the list is not completely built or it will be set to 1 if the list is completely built.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API. The list must currently be open. The list is opened by calling CWBSO_OpenList. If CWBSO_CloseList is called to close a list, CWBSO_OpenList must be called again before this API can be called.

CWBSO_GetObjAttr

Purpose: Retrieves the value of an attribute from an object.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_GetObjAttr(  
    CWBSO_OBJ_HANDLE objectHandle,  
    unsigned short usAttributeID,  
    char far* lpszBuffer,  
    unsigned long ulBufferLength,  
    unsigned long far* lpulBytesNeeded,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_OBJ_HANDLE objectHandle - input

A handle to an object that was returned by a previous call to CWBSO_GetObjHandle or CWBSO_CopyObjHandle.

unsigned short usAttributeID - input

The identifier of the attribute to be retrieved. The valid values for this parameter depend on the type of object. See the appropriate header files for the valid values:

- cwbsobj.h
- cwbsomsg.h
- cwbsopr.h
- cwbsosfl.h

char far* lpszBuffer - output

A long pointer to the output buffer where the attribute value will be placed. The value that is returned by this API is NOT a translated string. For instance, *END would be returned instead of Ending page for the ending page attribute of a spooled file. See “SOA attribute special values” on page 541 for information on special values that may be returned for each type of object. The output buffer is not changed when the return code is not set to CWBSO_NO_ERROR.

unsigned long ulBufferLength - input

The size, in bytes, of the output buffer argument.

unsigned long far* lpulBytesNeeded - output

A long pointer to an unsigned long that will be set to the number of bytes needed to place the entire attribute value in the output buffer. When this value is less than or equal to the size of output buffer that is specified, the entire attribute value is placed in the output buffer. When this value is greater than the size of output buffer that is specified, the output buffer contains a null string. The output buffer is not changed beyond the bytes that are needed for the attribute value. This value is set to zero when the return code is not set to CWBSO_NO_ERROR.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_OBJ_HANDLE

The object handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_BAD_ATTRIBUTE_ID

The attribute key is not valid for this object.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be called prior to calling this API. The object handle that is returned by CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API.

CWBSO_GetObjHandle

Purpose: Gets a handle to an object in a list. The object handle that is returned by this API is valid until the list is closed or until the object handle is deleted. The object handle may be used to call the following APIs:

- CWBSO_CopyObjHandle
- CWBSO_DeleteObjHandle
- CWBSO_DisplayObjAttr
- CWBSO_GetObjAttr
- CWBSO_RefreshObj
- CWBSO_SetObjAttr
- CWBSO_WaitForObj

Syntax:

```
unsigned int CWB_ENTRY CWBSO_GetObjHandle(  
    CWBSO_LIST_HANDLE listHandle,  
    unsigned long ulPosition,  
    CWBSO_OBJ_HANDLE far* lpObjectHandle,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that is returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

unsigned long ulPosition - input

The position of the object within the list for which a handle is needed. NOTE: The first object in a list is considered position 0.

CWBSO_OBJ_HANDLE far* lpObjectHandle - output

A long pointer to a handle which will be set to the handle for the iSeries object. This handle may be used with any other API that accepts an object handle with the exception that some APIs only operate on specific types of objects.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_BAD_LIST_POSITION

The position in list that is specified is not valid.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be

called prior to calling this API. The error handle that is returned by `CWBSO_CreateErrorHandle` must be passed as input to this API. The list must currently be open. The list is opened by calling `CWBSO_OpenList`. If `CWBSO_CloseList` is called to close a list, `CWBSO_OpenList` must be called again before this API can be called. You cannot access an object by using this API until that object has been included in the list. For example, if you issue this API to get the object in position 100 immediately after calling `CWBSO_OpenList`, the object may not immediately be available. In such instances, use `CWBSO_WaitForObj` to wait until an object is available. The object handle that is returned by this API must be deleted by a subsequent call to `CWBSO_DeleteObjHandle`.

CWBSO_OpenList

Purpose: Opens the list. A request is sent to the iSeries system to build the list.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_OpenList(  
    CWBSO_LIST_HANDLE listHandle,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error that was returned by a previous call to CWBSO_CreateErrorHandle. When the value that is returned by this API is CWBSO_ERROR_OCCURRED, the error handle may be used to retrieve the error message text or display the error to the user.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use the error for more information.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API. When the list is no longer needed, the calling program is responsible for doing the following:

- Call CWBSO_CloseList to close the list and free up resources that are allocated on the iSeries server.
- Call CWBSO_DeleteListHandle to free up resources that are allocated on the client.

CWBSO_ReadListProfile

Purpose: Reads the filter information for the list from the Windows Registry. The application name must have been set using the CWBSO_SetListProfile API. This API should be called prior to opening the list by using the CWBSO_OpenList or CWBSO_DisplayList APIs.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_ReadListProfile(  
    CWBSO_LIST_HANDLE listHandle,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object that was created by a previous call to CWBSO_CreateErrorHandle. When the value that is returned by this API is CWBSO_ERROR_OCCURRED, the error handle may be used to retrieve the error message text or display the error to the user.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_SYSTEM_NAME_DEFAULTED

No system name was specified on the CWBSO_CreateListHandle call for the list.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use the error handle for more information.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_SetListProfile must be called prior to calling this API. This API has no effect on a list that has been opened. In order for the filter criteria in the profile to take effect, the list must be opened after calling this API.

CWBSO_RefreshObj

Purpose: Refreshes an object's attributes from the iSeries server. Refreshes all open System Object Access views of the object.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_RefreshObj(  
    CWBSO_OBJ_HANDLE objectHandle,  
    HWND hWnd ,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_OBJ_HANDLE objectHandle - input

A handle to an object that was returned by a previous call to CWBSO_GetObjHandle or CWBSO_CopyObjHandle.

HWND hWnd - input

Handle of window to receive the focus after the refresh is complete. This parameter may be NULL. If this API is being called from an application window procedure, then the current window handle should be supplied. Otherwise, focus will shift to the most recently opened System Object Access window if one is open.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_OBJ_HANDLE

The object handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be called prior to calling this API. The object handle that is returned by CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API.

CWBSO_ResetParmObj

Purpose: Resets a parameter object to remove any attribute values from the object.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_ResetParmObj(  
    CWBSO_PARMOBJ_HANDLE parmObjHandle);
```

Parameters:

CWBSO_PARMOBJ_HANDLE parmObjHandle - input

A handle to a parameter object that was returned by a previous call to CWBSO_CreateParmObjHandle.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_PARMOBJ_HANDLE

The parameter object handle is not valid.

Usage: CWBSO_CreateParmObjHandle must be called prior to calling this API. The parameter object handle that is returned by CWBSO_CreateParmObjHandle must be passed as input to this API.

CWBSO_SetListFilter

Purpose: Sets a filter value for a list. Depending on the type of list, various filter values may be set. The filter values control which objects will be included in the list when the list is built by a call to CWBSO_OpenList.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_SetListFilter(  
    CWBSO_LIST_HANDLE listHandle,  
    unsigned short usFilterID,  
    char far* lpszValue);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

unsigned short usFilterID - input

The filter identifier specifies which portion of the filter to set. The valid values for this parameter depend on the type of objects in the list. See the appropriate header files for the valid values:

- cwbsobj.h
- cwbsomsg.h
- cwbsopr.h
- cwbsosfl.h

char far* lpszValue - input

The value for the filter attribute. If multiple items are specified, they must be separated by commas. Filter value items that specify iSeries object names must be in uppercase. Qualified object names must be in the form of library/object. Qualified job names must be in the form of job-number/user/job-name. Filter value items specifying special values (beginning with asterisk) must be specified in upper case. See “SOA attribute special values” on page 541 for information on the special values that may be supplied for each type of object.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_FILTER_ID

The filter ID specified is not valid for the type of list.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. This API has no effect on a list that has been opened. In order for the filter criteria to take effect, the list must be opened after calling this API. Caution should be used when requesting complex filters as list performance may be adversely affected.

CWBSO_SetListProfile

Purpose: Sets the profile name by adding the application name into the Windows Registry. Use CWBSO_ReadListProfile to read the filter information from the Registry prior to displaying a list. Use CWBSO_WriteListProfile to write the updated filter information to the Registry before deleting the list. If this API is not called, CWBSO_ReadListProfile and CWBSO_WriteListProfile will have no effect.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_SetListProfile(  
    CWBSO_LIST_HANDLE listHandle,  
    char far* lpszKey);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or to CWBSO_CreateListHandleEx.

char far* lpszKey - input

A long pointer to a string that will be used as the key in the Windows Registry for the list. This name could be the name of the application.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_PROFILE_NAME

The profile name that is specified is not valid.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API.

CWBSO_SetListSortFields

Purpose: Sets the sort criteria for a list. The sort criteria determines the order objects will appear in the list when the list is built by a call to CWBSO_OpenList. This API is only valid for lists of jobs and lists of spooled files. This API is not allowed for lists of messages and lists of printers.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_SetListSortFields(  
    CWBSO_LIST_HANDLE listHandle,  
    unsigned short far* lpusSortIDs,  
    unsigned short usCount);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

unsigned short far* lpusSortIDs - input

A long pointer to an array of sort column identifiers. The sort IDs specified will replace the current sort criteria for the list. The valid values for this parameter depend on the type of objects in the list. See the appropriate header files for the valid values:

- cwbsjob.h
- cwbsosfl.h

Note: If multiple sort IDs are specified, the order in which they appear in the array defines the order in which sorting will take place.

unsigned short usCount - input

The number of sort column identifiers specified.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_SORT_ID

A sort ID specified is not valid for the type of list.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_SORT_NOT_ALLOWED

Sorting is not allowed for this type of list.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. This API has no effect on a list that has been opened. In order for the sort criteria to take effect, the list must be opened after calling this API. Caution should be used when requesting complex sorts as list performance may be adversely affected.

CWBSO_SetListTitle

Purpose: Sets the title for a list. The title is displayed in the title bar of the window when the list is displayed by a call to CWBSO_DisplayList.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_SetListTitle(  
    CWBSO_LIST_HANDLE listHandle ,  
    char far* lpszTitle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

char far* lpszTitle - input

A long pointer to a string to be used for the list title. The length of the string must be less than or equal to 79.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_TITLE

The title that is specified is not valid.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API.

CWBSO_SetObjAttr

Purpose: Sets the value of one or more attributes of an object.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_SetObjAttr(  
    CWBSO_OBJ_HANDLE objectHandle,  
    CWBSO_PARMOBJ_HANDLE parmObjHandle,  
    unsigned short far* lpusErrorIndex,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_OBJ_HANDLE objectHandle - input

A handle to an object that was returned by a previous call to CWBSO_GetObjHandle or CWBSO_CopyObjHandle.

CWBSO_PARMOBJ_HANDLE parmObjHandle - input

A handle to a parameter object that was returned by a previous call to CWBSO_CreateParmObjHandle. The parameter object contains the attributes that are to be changed for the object.

unsigned short far* lpusErrorIndex - output

If an error occurred, this value will be set to the index of the parameter item that caused the error. The first parameter item is 1. This value will be set to 0 if none of the parameter items were in error.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_OBJECT_HANDLE

The object handle that is specified is not valid.

CWBSO_BAD_PARMOBJ_HANDLE

The parameter object handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_CANNOT_CHANGE_ATTRIBUTE

Attribute is not changeable at this time.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be called prior to calling this API. The object handle that is returned by CWBSO_GetObjHandle or CWBSO_CopyObjHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API.

CWBSO_SetParameter

Purpose: Sets the value of an attribute of an object. Multiple calls may be made to this API prior to calling CWBSO_SetObjAttr. This allows you to change several attributes for a specific object with one call to CWBSO_SetObjAttr.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_SetParameter(  
    CWBSO_PARMOBJ_HANDLE parmObjHandle,  
    unsigned short usAttributeID,  
    char far* lpszValue,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_PARMOBJ_HANDLE parmObjHandle - input

A handle to a parameter object that was returned by a previous call to CWBSO_CreateParmObjHandle.

unsigned short usAttributeID - input

The attribute ID for the parameter to be set. The valid values for this parameter depend on the type of object. See the appropriate header files for the valid values:

- cwbsobj.h
- cwbsomsg.h
- cwbsoprt.h
- cwbsosfl.h

char far* lpszValue - input

A long pointer to an attribute value. Note that only ASCII strings are accepted. Binary values must be converted to strings by using the appropriate library function. See "SOA attribute special values" on page 541 for information on the special values that may be supplied for each type of object.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_PARMOBJ_HANDLE

The parameter object handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_CreateParmObjHandle must be called prior to calling this API. The parameter object handle that is returned by CWBSO_CreateParmObjHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API. Calling this API does NOT update an object's attributes on the iSeries server. You must call CWBSO_SetObjAttr to actually update the attribute value or values on the iSeries server for the specified object.

CWBSO_WaitForObj

Purpose: Waits until an object is available in a list that is being built asynchronously.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_WaitForObj(  
    CWBSO_LIST_HANDLE listHandle,  
    unsigned long ulPosition,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

unsigned long ulPosition - input

The position of the desired object within the list. NOTE: The first object in a list is considered position 0.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object. If an error occurs that there is error text for, this handle may be used to retrieve the error message and message help.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_BAD_LIST_POSITION

The position in list that is specified does not exist.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use error handle for more information.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_CreateErrorHandle must be called prior to calling this API. The error handle that is returned by CWBSO_CreateErrorHandle must be passed as input to this API.

CWBSO_WriteListProfile

Purpose: Writes the filter information for the list to the specified key in the Windows registry. The key name must previously have been set using the CWBSO_SetListProfile API. This API should be called before deleting the list. This will save any filter criteria that was changed by the user during the CWBSO_DisplayList API. Filter information is saved in the registry by iSeries system and by type of list. For example, if your application accesses objects from two different iSeries systems, and displays all four types of lists, you would have eight different sections in the registry that specify filter information.

Syntax:

```
unsigned int CWB_ENTRY CWBSO_WriteListProfile(  
    CWBSO_LIST_HANDLE listHandle,  
    CWBSO_ERR_HANDLE errorHandle);
```

Parameters:

CWBSO_LIST_HANDLE listHandle - input

A handle to a list that was returned by a previous call to CWBSO_CreateListHandle or CWBSO_CreateListHandleEx.

CWBSO_ERR_HANDLE errorHandle - input

A handle to an error object that was created by a previous call to CWBSO_CreateErrorHandle. When the value that is returned by this API is CWBSO_ERROR_OCCURRED, the error handle may be used to retrieve the error message text or display the error to the user.

Return Codes: The following list shows common return values.

CWBSO_NO_ERROR

No error occurred.

CWBSO_BAD_LIST_HANDLE

The list handle that is specified is not valid.

CWBSO_BAD_ERR_HANDLE

The error handle that is specified is not valid.

CWBSO_SYSTEM_NAME_DEFAULTED

No system name was specified on the CWBSO_CreateListHandle call for the list.

CWBSO_LOW_MEMORY

Not enough memory is available for the request.

CWBSO_ERROR_OCCURRED

An error occurred. Use the error for more information.

Usage: CWBSO_CreateListHandle must be called prior to calling this API. The list handle that is returned by CWBSO_CreateListHandle must be passed as input to this API. CWBSO_SetListProfile must be called prior to calling this API.

SOA attribute special values

The topics that are listed below provide:

- A description of the special values that may be returned by **CWBSO_GetObjAttr**, and specified on **CWBSO_SetObjAttr**, for each type of object
- Any special values that may be specified on **CWBSO_SetListFilter** for each type of list object

Special considerations:

- For attributes that are numeric, it is common practice for iSeries APIs to return negative numeric values to indicate which special value (if any) an object attribute contains. System Object Access automatically maps these negative numbers to their corresponding special value string. For example, the Retrieve Spooled File Attributes (QUSRSPLA) API returns "-1" for page rotation if output reduction is performed automatically. **CWBSO_GetObjAttr** returns "**AUTO".
- Some list filter criteria accept multiple values. For example, it is possible to filter a list of printers on multiple printer names. In such cases, commas should separate the supplied values.

Where to find additional information about attribute special values:

See the OS/400 APIs topic in the iSeries Information Center.

SOA attribute special values:

- "Job attributes"
- "Message attributes"
- "Printer attributes" on page 542
- "Printer output attributes" on page 547
- "TCP/IP interfaces attributes" on page 547
- "Ethernet lines attributes" on page 548
- "Token-ring lines attributes" on page 549
- "Hardware resources attributes" on page 549
- "Software products attributes" on page 549
- "TCP/IP routes attributes" on page 549
- "Users and groups attributes" on page 550
- "Libraries in QSYS attributes" on page 553

Job attributes: System Object Access uses the **List Job** (QUSLJOB) and **Retrieve Job Information** (QUSRJOBI) iSeries APIs to retrieve attributes for jobs. The possible special values are the same as those that are documented in the OS/400 APIs: Work Management APIs topic in the iSeries Information Center. The following special value mappings are not documented explicitly:

CWBSO_JOB_CpuTimeUsed

If the field is not large enough to hold the actual result, QUSRJOBI returns -1. System Object Access returns "++++".

CWBSO_JOB_MaxCpuTimeUsed,

CWBSO_JOB_MaxTemporaryStorage,

CWBSO_JOB_DefaultWaitTime

If the value is *NOMAX, QUSRJOBI returns -1. System Object Access returns "**NOMAX".

CWBSO_SetListFilter accepts all special values that are supported by the List Job (QUSLJOB) API.

Message attributes: System Object Access uses the **List Nonprogram Messages** (QMHLSTM) OS/400 API to retrieve attributes for messages. The possible special values are the same as those that are documented in the OS/400 APIs: Message Handling APIs topic in the iSeries Information Center.

CWBSO_SetListFilter accepts the special values that are supported by the List Nonprogram Messages (QMHLSTM) API for Severity Criteria. In addition, a 10-character user name may be supplied, by specifying the **CWBSO_MSGF_UserName** filter ID. “CURRENT” may be used to obtain a list of messages for the current user.

Printer attributes: System Object Access uses undocumented iSeries APIs to retrieve attributes for printer objects. A printer is a “logical” object that is actually a combination of a device description, a writer, and an output queue. The attributes and their possible values are as follows.

CWBSO_PRT_AdvancedFunctionPrinting. Whether the printer device supports Advanced Function Printing (AFP).

***NO** The printer device does not support Advanced Function Printing.

***YES** The printer device supports Advanced Function Printing.

CWBSO_PRT_AllowDirectPrinting. Whether the printer writer allows the printer to be allocated to a job that prints directly to a printer.

***NO** Direct printing is not allowed

***YES** Direct printing is allowed.

CWBSO_PRT_BetweenCopiesStatus. Whether the writer is between copies of a multiple copy spooled file. The possible values are Y (yes) or N (no).

CWBSO_PRT_BetweenFilesStatus. Whether the writer is between spooled files. The possible values are Y (yes) or N (no).

CWBSO_PRT_ChangesTakeEffect. The time at which the pending changes to the writer take effect. Possible values are:

***NORDYF**

When all the current eligible files are printed.

***FILEEND**

When the current spooled file is done printing.

blank No pending changes to the writer.

CWBSO_PRT_CopiesLeftToProduce. The number of copies that are left to be printed. This field is set to 0 when no file is printing.

CWBSO_PRT_CurrentPage. The page number in the spooled file that the writer is currently processing. The page number shown may be lower or higher than the actual page number being printed because of buffering done by the system. This field is set to 0 when no spooled file is printing.

CWBSO_PRT_Description. The text description of the printer device.

CWBSO_PRT_DeviceName. The name of the printer device.

CWBSO_PRT_DeviceStatus. The status of the printer device. Possible values are the same as the device status that is returned by the Retrieve Configuration Status (QDCRCFGS) API.

CWBSO_PRT_EndAutomatically. When to end the writer if it is to end automatically.

***NORDYF**

When no files are ready to print on the output queue from which the writer is selecting files to be printed.

***FILEEND**

When the current spooled file has been printed.

***NO** The writer will not end, but it will wait for more spooled files.

CWBSO_PRT_EndPendingStatus. Whether an End Writer (ENDWTR) command has been issued for this writer. Possible values are:

N No ENDWTR command was issued.

- I** *IMMED: The writer ends as soon as its output buffers are empty.
- C** *CNTRLD: The writer ends after the current copy of the spooled file has been printed.
- P** *PAGEEND: The writer ends at the end of the page.

CWBSO_PRT_FileName. The name of the spooled file that the writer is currently processing. This field is blank when no file is printing.

CWBSO_PRT_FileNumber. The number of the spooled file that the writer is currently processing. This field is set to 0 when no spooled file is printing.

CWBSO_PRT_FormsAlignment. The time at which the forms alignment message will be sent. Possible values are:

***WTR** The writer determines when the message is sent.

***FILE** Control of the page alignment is specified by each file.

CWBSO_PRT_FormType. The type of form that is being used to print the spooled file. Possible values are:

***ALL** The writer is started with the option to print all spooled files of any form type.

***FORMS**

The writer is started with the option to print all the spooled files with the same form type before using a different form type.

***STD** The writer is started with the option to print all the spooled files with a form type of *STD.

form type name

The writer is started with the option to print all the spooled files with the form type you specified.

CWBSO_PRT_FormTypeNotification. Message option for sending a message to the message queue when this form is finished. Possible values are:

***MSG** A message is sent to the message queue.

***NOMSG**

No message is sent to the message queue.

***INFOMSG**

An informational message is sent to the message queue.

***INQMSG**

An inquiry message is sent to the message queue.

CWBSO_PRT_HeldStatus. Whether the writer is held. The possible values are Y (yes) or N (no).

CWBSO_PRT_HoldPendingStatus. Whether a Hold Writer (HLDWTR) command has been issued for this writer. Possible values are:

N No HLDWTR command was issued.

I *IMMED: The writer is held as soon as its output buffers are empty.

C *CNTRLD: The writer is held after the current copy of the file has been printed.

P *PAGEEND: The writer is held at the end of the page.

CWBSO_PRT_JobName. The name of the job that created the spooled file which the writer is currently processing. This field is blank when no spooled file is printing.

CWBSO_PRT_JobNumber. The number of the job that created the spooled file which the writer currently is processing. This field is blank when no spooled file is printing.

CWBSO_PRT_MessageKey. The key to the message that the writer is waiting for a reply. This field will be blank when the writer is not waiting for a reply to an inquiry message.

CWBSO_PRT_MessageQueueLibrary. The name of the library that contains the message queue.

CWBSO_PRT_MessageQueueName. The name of the message queue that this writer uses for operational messages.

CWBSO_PRT_MessageWaitingStatus. Whether the writer is waiting for a reply to an inquiry message. The possible values are Y (yes) or N (no).

CWBSO_PRT_NextFormType. The name of the next form type to be printed. Possible values are:

***ALL** The writer is changed with the option to print all spooled files of any form type.

***FORMS**

The writer is changed with the option to print all the spooled files with the same form type before using a different form type.

***STD** The writer is changed with the option to print all the spooled files with a form type of *STD.

form type name

The writer is changed with the option to print all the spooled files with the form type name you specified.

blank No change has been made to this writer.

CWBSO_PRT_NextFormTypeNotification. The message option for sending a message to the message queue when the next form type is finished. Possible values are:

***MSG** A message is sent to the message queue.

***NOMSG**

No message is sent to the message queue.

***INFOMSG**

An informational message is sent to the message queue.

***INQMSG**

An inquiry message is sent to the message queue.

blank No change is pending.

CWBSO_PRT_NextOutputQueueLibrary. The name of the library that contains the next output queue. This field is blank if no changes have been made to the writer.

CWBSO_PRT_NextOutputQueueName. The name of the next output queue to be processed. This field is blank if no changes have been made to the writer.

CWBSO_PRT_NextSeparatorDrawer. This value indicates the drawer from which to take the separator pages if there is a change to the writer. Possible values are:

***FILE** Separator pages print from the same drawer that the spooled file prints from. If you specify a drawer different from the spooled file that contains colored or different type paper, the page separator is more identifiable.

***DEV** Separator pages print from the separator drawer that is specified in the printer device description.

empty string

No pending change to the writer.

1 The first drawer.

2 The second drawer.

3 The third drawer.

CWBSO_PRT_NextSeparators. The next number of separator pages to be printed when the change to the writer takes place. Possible values are:

***FILE** The number of separator pages is specified by each file.

empty string

No pending change to the writer.

number of separators

The number of separator pages to be printed.

CWBSO_PRT_NumberOfSeparators. The number of separator pages to be printed. Possible values are:

***FILE** The number of separator pages is specified by each file.

Number of separators

The number of separator pages to be printed.

CWBSO_PRT_OnJobQueueStatus. Whether the writer is on a job queue and, therefore, is not currently running. The possible values are Y (yes) or N (no).

CWBSO_PRT_OutputQueueLibrary. The name of the library that contains the output queue from which spooled files are selected for printing.

CWBSO_PRT_OutputQueueName. The name of the output queue from which spooled files are being selected for printing.

CWBSO_PRT_OutputQueueStatus. The status of the output queue from which spooled files are being selected for printing. Possible values are:

H The output queue is held.

R The output queue is released.

CWBSO_PRT_PrinterDeviceType. The type of the printer that is being used to print the spooled file. Valid values are:

***SCS** SNA (Systems Network Architecture) character stream

***IPDS** Intelligent Printer Data Stream

CWBSO_PRT_SeparatorDrawer. Identifies the drawer from which the job and file separator pages are to be taken. Possible values are:

***FILE** The separator page prints from the same drawer that the file is printed from. If you specify a drawer different from the file that contains colored or different type paper, the page separator is more identifiable.

***DEVDD** The separator pages will print from the separator drawer that is specified in the printer device description.

1 The first drawer.

2 The second drawer.

3 The third drawer.

CWBSO_PRT_StartedByUser. The name of the user that started the writer.

CWBSO_PRT_Status. The overall status of the logical printer. This field is derived from the printer device status (from the Retrieve Configuration Status QDCRCFGS API), the output queue status (from the List Printer and Writer Status, SPLSTPRT, XPF macro) and writer status (from the Retrieve Writer Information, QSPRWTRI, API). Possible values are:

1 Unavailable

2 Powered off or not yet available

3 Stopped

4 Message waiting

5 Held

6 Stop (pending)

7 Hold (pending)

8 Waiting for printer

9 Waiting to start

10 Printing

11 Waiting for printer output

12 Connect pending

13 Powered off

- 14 Unusable
- 15 Being serviced
- 999 Unknown

CWBSO_PRT_TotalCopies. The total number of copies to be printed.

CWBSO_PRT_TotalPages. The total number of pages in the spooled file. Possible values are:

number

The number of pages in the spooled file.

0 No spooled file is printing.

CWBSO_PRT_User. The name of the user who created the spooled file that the writer is currently processing. This field is blank when no file is printing.

CWBSO_PRT_UserSpecifiedData. The user-specified data that describe the file that the writer is currently processing. This field is blank when no file is printing.

CWBSO_PRT_WaitingForDataStatus. Whether the writer has written all the data that is currently in the spooled file and is waiting for more data. Possible values are:

N The writer is not waiting for more data.

Y The writer has written all the data currently in the spooled file and is waiting for more data. This condition occurs when the writer is producing an open spooled file with SCHEDULE(*IMMED) that is specified.

CWBSO_PRT_WaitingForDeviceStatus. Whether the writer is waiting to get the device from a job that is printing directly to the printer.

N The writer is not waiting for the device.

Y The writer is waiting for the device

CWBSO_PRT_WriterJobName. The job name of the printer writer.

CWBSO_PRT_WriterJobNumber. The job number of the printer writer.

CWBSO_PRT_WriterJobUser. The name of the system user.

CWBSO_PRT_WriterStarted. Indication of whether a writer is started for this printer. Possible values are:

0 No writer is started

1 Writer is started

CWBSO_PRT_WriterStatus. The status of the writer for this printer. Possible values are:

X'01' Started

X'02' Ended

X'03' On job queue

X'04' Held

X'05' Waiting on message

CWBSO_PRT_WritingStatus. Whether the printer writer is in writing status. The possible values are:

Y The writer is in writing status.

N The writer is not in writing status.

S The writer is writing the file separators.

System Object Access accepts a comma-separated list of printer names. Up to 100 printer names may be specified. A special value of “*ALL” may be supplied to request a list of all printers on the iSeries server.

Printer output attributes: System Object Access uses the **List Spooled Files** (QUSLSPL) and **Retrieve Spooled File Attributes** (QUSRSPLA) iSeries APIs to retrieve attributes for printer output. The possible special values are the same as those that are documented in the OS/400 APIs: Spooled File APIs topic in the iSeries Information Center. The following special value mappings are not explicitly documented:

CWBSO_SFL_StartingPage

If the ending page value is to be used, QUSRSPLA returns -1. System Object Access returns “*ENDPAGE”.

CWBSO_SFL_EndingPage

If the last page is to be the ending page, QUSRSPLA returns 0 or 2147483647. System Object Access returns “*END”.

CWBSO_SFL_MaximumRecords

If there is no maximum, QUSRSPLA returns 0. System Object Access returns “*NOMAX”.

CWBSO_SFL_PageRotation

If no rotation is done, QUSRSPLA returns 0. System Object Access returns “*NONE”.

An undocumented API is used to retrieve the printer device name or names for a spooled file. The attribute and its possible values are described below.

CWBSO_SFL_DeviceNames. The name of the printer device that will print the file. If the printer output is assigned to more than one printer device, this field contains all of the printer names in the group of printers. Possible values are:

printer name

The name of the printer to which the printer output is assigned.

list of printer names

The names of the printers in the group to which the printer output is assigned. Commas will separate the printer names.

empty string

The printer output is not assigned to a printer or group of printers.

CWBSO_SetListFilter accepts all special values that are supported by the List Spooled Files (QUSLSPL) API.

TCP/IP interfaces attributes: System Object Access uses the iSeries API TCP/IP interface (QTOCIFCU) to retrieve attributes for TCP/IP interfaces. The possible special values are:

CWBSO_TIF_TCPIPNetworkName

CWBSO_TIF_InternetAddress

CWBSO_TIF_BinaryInternetAddress

*RTVIFCLST only - The list of interfaces returned immediately will follow the I/O Variable header. The interface structure will repeat for each interface returned.

CWBSO_TIF_SubnetMask

CWBSO_TIF_AssociatedLocalInterface

CWBSO_TIF_BinaryLocalIP

*RTVIFCLST only - The list of interfaces returned immediately will follow the I/O Variable header. The interface structure will repeat for each interface returned.

CWBSO_TIF_LineDescriptionName

CWBSO_TIF_TypeOfLine

- 1=Ethernet
- 2=Token ring
- 3=Frame relay

- 4=Asynchronous
- 5=PPP
- 6=Wireless
- 7=X.25
- 8=DDI
- 9=Twinaxial (TDLC)
- 15=None
- 16=Error
- 17=Not found

CWBSO_TIF_MaximumTransmissionUnit

- 0000000 = Determined by Maximum frame size in Line Description.

CWBSO_TIF_TypeOfService

- 1=Normal
- 2=Minimum delay
- 3=Maximum throughput
- 4=Maximum reliability
- 5=Minimum cost

CWBSO_TIF_AutomaticStart

- 1=Yes
- 2=No

CWBSO_TIF-TokenRingBitSequence

- 1=MSB
- 2=LSB

CWBSO_TIF_Status

*RTVIFCLST only - The list of interfaces returned immediately will follow the I/O Variable header. The interface structure will repeat for each interface returned.

- 0=Inactive
- 1=Active
- 2=Starting
- 3=Ending
- 4=Recovery pending
- 5=Recovery cancel
- 6=Failed
- 7=Failed (TCP)

CWBSO_TIF_InterfaceName

CWBSO_TIF_PPPProfile

*RTVIFCLST only - The list of interfaces returned immediately will follow the I/O Variable header. The interface structure will repeat for each interface returned.

CWBSO_TIF_PPPRemoteIP

*RTVIFCLST only - The list of interfaces returned immediately will follow the I/O Variable header. The interface structure will repeat for each interface returned.

CWBSO_TIF_ApplicationDefined

Ethernet lines attributes: See the OS/400 APIs: Configuration APIs topic in the iSeries Information Center.

Token-ring lines attributes: See the OS/400 APIs: Configuration APIs topic in the iSeries Information Center.

Hardware resources attributes: See the OS/400 APIs: Hardware Resource APIs topic in the iSeries Information Center.

Software products attributes: See the OS/400 APIs: Software Product APIs topic in the iSeries Information Center.

TCP/IP routes attributes: System Object Access uses the iSeries API TCP/IP route (QTOCRTEU) to retrieve attributes for TCP/IP routes. The possible special values are:

CWBSO_RTE_TCPIPNetworkName

CWBSO_RTE_InternetAddress

CWBSO_RTE_BinaryInternetAddress

*RTVxxxLST only - The list of routes returned immediately will follow the I/O Variable header. The interface structure will repeat for each route returned.

CWBSO_RTE_SubnetMask

CWBSO_RTE_BinarySubnetMask

*RTVxxxLST only - The list of routes returned immediately will follow the I/O Variable header. The interface structure will repeat for each route returned.

CWBSO_RTE_NextHopAddress

CWBSO_RTE_BinaryNextHop

*RTVxxxLST only - The list of routes returned immediately will follow the I/O Variable header. The interface structure will repeat for each route returned.

CWBSO_RTE_BindingInterface

CWBSO_RTE_BinaryBindingIP

*RTVxxxLST only - The list of routes returned immediately will follow the I/O Variable header. The interface structure will repeat for each route returned.

CWBSO_RTE_MaximumTransmissionUnit

CWBSO_RTE_TypeOfService

- 1=Normal
- 2=Minimum delay
- 3=Maximum throughput
- 4=Maximum reliability
- 5=Minimum cost

CWBSO_RTE_RoutePrecedence

CWBSO_RTE_RIPMetric

CWBSO_RTE_RIPRedistribution

- 1=Yes
- 2=No

CWBSO_RTE_PPPProfile

Not valid for *xxxRTE

CWBSO_RTE_PPPCallerUserid

Not valid for *xxxRTE

CWBSO_RTE_PPPCallerIP

Not valid for *xxxRTE

CWBSO_RTE_ApplicationDefined

Users and groups attributes: The possible users and groups special values are valid:

CWBSO_USR_ProfileName
CWBSO_USR_ProfileOrGroupIndicator
CWBSO_USR_GroupHasMembers
CWBSO_USR_TextDescription
CWBSO_USR_PreviousSignonDate
CWBSO_USR_PreviousSignonTime
CWBSO_USR_SignonAttemptsNotValid
CWBSO_USR_Status
CWBSO_USR_PasswordChangeDate
CWBSO_USR_NoPasswordIndicator
CWBSO_USR_PasswordExpirationInterval
CWBSO_USR_DatePasswordExpires
CWBSO_USR_DaysUntilPasswordExpires
CWBSO_USR_SetPasswordToExpire
CWBSO_USR_DisplaySignonInformation
CWBSO_USR_UserClassName
CWBSO_USR_AllObjectAccess
CWBSO_USR_SecurityAdministration
CWBSO_USR_JobControl
CWBSO_USR_SpoolControl
CWBSO_USR_SaveAndRestore
CWBSO_USR_SystemServiceAccess
CWBSO_USR_AuditingControl
CWBSO_USR_SystemConfiguration
CWBSO_USR_GroupProfileName
CWBSO_USR_Owner
CWBSO_USR_GroupAuthority
CWBSO_USR_LimitCapabilities
CWBSO_USR_GroupAuthorityType
CWBSO_USR_SupplementalGroups
CWBSO_USR_AssistanceLevel
CWBSO_USR_CurrentLibraryName
CWBSO_USR_InitialMenuName
CWBSO_USR_InitialMenuLibraryName
CWBSO_USR_InitialProgramName
CWBSO_USR_InitialProgramLibraryName
CWBSO_USR_LimitDeviceSessions
CWBSO_USR_KeyboardBuffering
CWBSO_USR_MaximumAllowedStorage
CWBSO_USR_StorageUsed
CWBSO_USR_HighestSchedulingPriority
CWBSO_USR_JobDescriptionName

CWBSO_USR_JobDescriptionNameLibrary
CWBSO_USR_AccountingCode
CWBSO_USR_MessageQueueName
CWBSO_USR_MessageQueueLibraryName
CWBSO_USR_MessageQueueDeliveryMethod
CWBSO_USR_MessageQueueSeverity
CWBSO_USR_OutputQueue
CWBSO_USR_OutputQueueLibrary
CWBSO_USR_PrintDevice
CWBSO_USR_SpecialEnvironment
CWBSO_USR_AttentionKeyHandlingProgramName
CWBSO_USR_AttentionKeyHandlingProgramLibrary
CWBSO_USR_LanguageID
CWBSO_USR_CountryID
CWBSO_USR_CharacterCodeSetID
CWBSO_USR_ShowParameterKeywords
CWBSO_USR_ShowAllDetails
CWBSO_USR_DisplayHelpOnFullScreen
CWBSO_USR_ShowStatusMessages
CWBSO_USR_DoNotShowStatusMessages
CWBSO_USR_ChangeDirectionOfRollkey
CWBSO_USR_SendMessageToSpoolFileOwner
CWBSO_USR_SortSequenceTableName
CWBSO_USR_SortSequenceTableLibraryName
CWBSO_USR_DigitalCertificateIndicator
CWBSO_USR_CharacterIDControl
CWBSO_USR_ObjectAuditValue
CWBSO_USR_CommandUsage
CWBSO_USR_ObjectCreation
CWBSO_USR_ObjectDeletion
CWBSO_USR_JobTasks
CWBSO_USR_ObjectManagement
CWBSO_USR_OfficeTasks
CWBSO_USR_ProgramAdoption
CWBSO_USR_SaveAndRestoreTasks
CWBSO_USR_SecurityTasks
CWBSO_USR_ServiceTasks
CWBSO_USR_SpoolManagement
CWBSO_USR_SystemManagement
CWBSO_USR_OpticalTasks
CWBSO_USR_UserIDNumber
CWBSO_USR_GroupIDNumber
CWBSO_USR_DoNotSetAnyJobAttributes
CWBSO_USR_UseSystemValue
CWBSO_USR_CodedCharacterSetID

CWBSO_USR_DateFormat
CWBSO_USR_DateSeparator
CWBSO_USR_SortSequenceTable
CWBSO_USR_TimeSeparator
CWBSO_USR_DecimalFormat
CWBSO_USR_HomeDirectoryDelimiter
CWBSO_USR_HomeDirectory
CWBSO_USR_Locale
CWBSO_USR_IndirectUser
CWBSO_USR_PrintCoverPage
CWBSO_USR_MailNotification
CWBSO_USR_UserID
CWBSO_USR_LocalDataIndicator
CWBSO_USR_UserAddress
CWBSO_USR_SystemName
CWBSO_USR_SystemGroup
CWBSO_USR_UserDescription
CWBSO_USR_FirstName
CWBSO_USR_PREFERREDNAME
CWBSO_USR_MiddleName
CWBSO_USR_LastName
CWBSO_USR_FullName
CWBSO_USR_JobTitle
CWBSO_USR_CompanyName
CWBSO_USR_DepartmentName
CWBSO_USR_NetworkUserID
CWBSO_USR_PrimaryTelephoneNumber
CWBSO_USR_SecondaryTelephoneNumber
CWBSO_USR_FaxNumber
CWBSO_USR_Location
CWBSO_USR_BuildingNumber
CWBSO_USR_OfficeNumber
CWBSO_USR_MailingAddress
CWBSO_USR_MailingAddress2
CWBSO_USR_MailingAddress3
CWBSO_USR_MailingAddress4
CWBSO_USR_CCMailAddress
CWBSO_USR_CCMailComment
CWBSO_USR_MailServerFrameworkServiceLevel
CWBSO_USR_PREFERREDADDRESSFIELDNAME
CWBSO_USR_PREFERREDADDRESSPRODUCTID
CWBSO_USR_PREFERREDADDRESSTYPEVALUE
CWBSO_USR_PREFERREDADDRESSTYPENAME
CWBSO_USR_PREFERREDADDRESS
CWBSO_USR_ManagerCode

CWBSO_USR_SMTPUserID
CWBSO_USR_SMTPDomain
CWBSO_USR_SMTPRoute
CWBSO_USR_GroupMemberIndicator

Note: In release/version V4R4 and later, the following attributes are meaningful only when Lotus Notes is installed on the iSeries server:

CWBSO_USR_NotesServerName
CWBSO_USR_NotesCertifierID
CWBSO_USR_MailType
CWBSO_USR_NotesMailFileName
CWBSO_USR_CreateMailFiles
CWBSO_USR_NotesForwardingAddress
CWBSO_USR_SecurityType
CWBSO_USR_LicenseType
CWBSO_USR_MinimumNotesPasswordLength
CWBSO_USR_UpdateExistingNotesUser
CWBSO_USR_NotesMailServer
CWBSO_USR_LocationWhereUserIDsStored
CWBSO_USR_ReplaceExistingNotesID
CWBSO_USR_NotesComment
CWBSO_USR_NotesUserLocation
CWBSO_USR_UserPassword
CWBSO_USR_NotesUserPassword
CWBSO_USR_NotesCertifierPassword
CWBSO_USR_ShortName

Libraries in QSYS attributes: See the OS/400 APIs: Object APIs topic in the iSeries Information Center.

Chapter 5. iSeries Access for Windows Database Programming

iSeries Access for Windows provides multiple programming interfaces for accessing database files on the iSeries server. Some of these interfaces are common interfaces, allowing you to write a single application that accesses both the iSeries database and non-iSeries databases. iSeries Access for Windows also supports a proprietary C API to expose the unique strengths of DB2® for iSeries.

iSeries Access for Windows provides both Structured Query Language (SQL) and record-level access interfaces. The SQL interfaces provide access to DB2 Universal Database (UDB) for iSeries database files and stored procedures. The record-level access interface provides the fastest access to single records within a file. The IBM *DB2 UDB for iSeries SQL Programming* book contains detailed information.

How to access the book:

Follow these steps to view a hypertext markup language (HTML) online version of the *DB2 UDB for iSeries SQL Programming* book, and to print a PDF version:

1. Link to the DB2 Universal Database™ for iSeries books online topic in the **iSeries Information Center**
2. Select **SQL Programming Concepts**

| The following iSeries Access for Windows database interfaces expose C/C++ interfaces, however, knowing C/C++ is not a requirement:

| **“iSeries Access for Windows OLE DB Provider”**

| Supports record-level access and SQL access to iSeries database files. Use the ActiveX Data Objects (ADO) and the OLE DB interfaces to take advantage of this support.

| **“iSeries Access for Windows ODBC” on page 556**

| A common database interface that uses SQL as its database access language. iSeries Access for Windows provides an ODBC driver to support this interface.

| **“iSeries Access for Windows database APIs” on page 646**

| The iSeries Access for Windows proprietary C/C++ Database APIs provide support for iSeries database and catalog functions, in addition to SQL access to iSeries database files.

| **Note:** Read the Chapter 1, “Code disclaimer information” on page 3 for important legal information.

iSeries Access for Windows OLE DB Provider

The iSeries Access for Windows OLE DB Provider, along with the Programmer’s Toolkit, makes iSeries client/server application development quick and easy from the Windows client PC. The iSeries Access for Windows OLE DB Provider gives iSeries programmers record-level access interfaces to iSeries logical and physical DB2 Universal Database (UDB) for iSeries database files. In addition, it provides support for SQL, data queues, programs, and commands. If you use Visual Basic, the Visual Basic Wizards make it simple and easy to develop customized, working applications.

ADO and OLE DB standards provide programmers with consistent interfaces to iSeries server data and services. The IBMADA400 Provider handles all iSeries server-to-PC and data type-to-data type conversions.

To install OLE DB Provider:

When you install iSeries Access for Windows (or when you run **Selective Setup** if iSeries Access for Windows is installed), select the **Data Access** component. Make sure that the **OLE DB Provider** subcomponent also is selected.

Note: The OLE DB Provider will not be installed if the computer does not have MDAC 2.5 or later installed, before installing iSeries Access for Windows. MDAC can be downloaded from Microsoft: <http://www.microsoft.com/data/doc.htm>.



To access OLE DB Technical Reference:

The iSeries Access for Windows OLE DB Technical Reference, which is shipped with iSeries Access for Windows, provides complete documentation of OLE DB Provider support. To access it from the Programmer's Toolkit, select **Overview** → **Common Interfaces** → **ADO/OLE DB**.

To install Programmer's Toolkit and the iSeries ADO Wizards for Visual Basic:

When you install iSeries Access for Windows (or when you run **Selective Setup** if iSeries Access for Windows is installed), select the **Programmer's Toolkit** component. See "Installing the Programmer's Toolkit" on page 12 for more information.

Other OLE DB information resources:

- IBM iSeries Access for Windows OLE DB Support Web site. 
- IBM Redbook Fast Path to iSeries Client/Server Using iSeries OLE DB Support: SG24-5183 

iSeries Access for Windows ODBC

What is ODBC?

ODBC stands for open database connectivity. It consists of:

- A well-defined set of functions (application programming interfaces)
- Standards for SQL syntax (that are recommended but not imposed)
- Error codes
- Data types

The application programming interfaces provide a rich set of functions to connect to a database management system, run SQL statements and to retrieve data. Also included are functions to interrogate the SQL catalog of the database and the capabilities of the driver.

ODBC drivers return standard error codes and translate data types to a common (ODBC) standard. ODBC allows the application developer to obtain integrated database error information, and to avoid some of the most complex problems that are involved with making applications portable.

What you can do with ODBC:

Use ODBC to:

- Send SQL requests to the database management system (DBMS).
- Use the same program to access different database management system (DBMS) products without re-compiling.
- Create an application that is independent of the data communications protocol.
- Handle data in a format convenient to the application.

The flexibility of ODBC APIs allows you to use them in transaction-based, line-of-business applications (where the SQL is predefined). They also can use them in query tools such as Lotus® Approach® or Microsoft Query (where the select statement is created at run time).

Structured Query Language (SQL):

ODBC supports dynamic SQL, which sometimes is associated with poor performance. However, careful use of parameter markers enables repeated statements to achieve static SQL-like performance. Also, extended dynamic SQL—a special capability of the iSeries Access for Windows ODBC driver—enables previously prepared SQL statements to achieve performance that rivals static SQL.

Where to find information on SQL:


For more information on SQL, see the IBM *SQL Reference* book. View an HTML online version of the book, or print a PDF version, from the DB2 Universal Database for iSeries books online iSeries Information Center topic.

iSeries Access for Windows ODBC topics:

Note: The information linked to from this page applies to the iSeries Access for Windows 32-bit ODBC driver, the iSeries Access for Windows 64-bit ODBC driver, and the iSeries Access for Linux ODBC driver. For additional information regarding setup for the iSeries Access for Linux ODBC driver see iSeries ODBC Driver for Linux.

- “Implementation issues of ODBC APIs” on page 586
- “ODBC API restrictions and unsupported functions” on page 597
- “ODBC APIs”
- **ODBC 3.x APIs**
- “iSeries Access for Windows ODBC performance” on page 602
- “ODBC programming examples” on page 640

Where to find documentation on the ODBC standard:

See the Microsoft ODBC Web site 

ODBC APIs

ODBC APIs required files:

Header files	Import library	Dynamic Link Library
sql.h	odbc32.lib	odbc32.dll
sqlext.h		
sqltypes.h		
sqlucode.h		

Programmer's Toolkit:

The Programmer's Toolkit provides ODBC documentation, and links to sample programs and related information. To access this information, open the Programmer's Toolkit and select **Database → ODBC**.

ODBC APIs topics:

- **iSeries Access for Windows ODBC APIs listing**
- “Implementation issues of ODBC APIs” on page 586

ODBC APIs: General concepts

The following general concepts apply to ODBC APIs:

Environments:

The environment in which Windows makes available some memory for ODBC to monitor its run-time information.

Connections:

Within the environment there can be multiple connections, each to a data source. The connections may be to different physical servers, to the same server, or any combination of both.

Statements:

Multiple statements can be run within each connection.

Handles:

Handles are identifiers for storage areas that are allocated by the Driver Manager or individual drivers. The three types of handles are:

Environment handle:

Global information, that includes other handles. One handle is allowed per application.

Connection handle:

Information about connection to a data source. Multiple connection handles are allowed per environment.

Statement handle:

Information about a particular SQL statement. Multiple statement handles are allowed per connection. Statement handles can be reused for other SQL statements and long as the statement state is valid.

Descriptor handle:

Information about explicit descriptors that are associated with the connection handle. The application creates these, and asks the driver to use them instead of the implicit descriptors associated with a statement handle.

Essentially, a **handle** can be considered as an identifier for a resource that is recognized by ODBC (an environment, connection or statement). ODBC provides an identifier (the handle) for this resource that you can use in your program. Exactly what ODBC stores in the handle (which is held as a long integer) is not relevant. Be careful not to change the value, and to assign unique names to the variables that hold the various handles.

Some APIs set the handle (for example, `SQLAllocEnv` or `SQLAllocHandle` with `SQL_HANDLE_ENV` handle type), and you must pass in a reference, or pointer to the variable. Some APIs refer to a handle that previously was set (for example, **SQLExecute**), and you must pass in the variable by value.

ODBC 3.x APIs

The following table lists ODBC 3.x APIs by their associated task and identifies considerations for each API. For some global API considerations, see “ODBC API restrictions and unsupported functions” on page 597. For other implementation issues and related topics see “Implementation issues of ODBC APIs” on page 586.

Note: The iSeries Access for Windows ODBC Driver is a Unicode driver; however, ANSI applications will still continue to work with it. The ODBC Driver Manager will handle converting an ANSI ODBC API call to the wide version before calling the iSeries Access for Windows ODBC Driver. To write a Unicode application, you must call the wide version for some of these APIs. When writing an application to the wide ODBC interface, you need to know whether the length for each API is defined as character, in bytes, or if the length is not applicable. Refer to the 'Type' column in the following table for this information.

Attribute	Type	API	Description	Other considerations
Connecting to a data source	N/A	SQLAllocHandle	Obtains an environment and connection handle. One environment handle is used for one or more connections. May also allocate a statement or a descriptor handle.	

	Char	SQLConnect	Connects to a specific data source name with a specific user ID and password.	There is an option to control whether this API prompts a signon dialog when the user ID and password are not specified. This option can be set from the Connection options dialog on the General tab of the DSN. See "Signon dialog behavior" on page 597 for more information.
	Char	SQLDriverConnect	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialogs for the user.	<p>Uses all keywords. Only DSN is required. Other values are optional. Refer to "Connection String keywords" on page 586 and the iSeries Access for Windows Users' Guide for more information.</p> <p>Only 25 libraries are supported in a library list on a connection to a pre-V5R1 server. 75 entries are supported on a V5R1 and later servers. Entries over 75 are ignored.</p> <p>For information on how this API prompts signon dialogs see "Signon dialog behavior" on page 597.</p>
	Char	SQLBrowseConnect	Returns successive levels of connection attributes and valid attribute values. When a value has been specified for each connection attribute, connects to the data source.	To make a connection attempt the SYSTEM keyword and either the DSN or DRIVER keywords must be specified. All the other keywords are optional. Note, the PWD keyword is not returned in the output string for security purposes. Refer to "Signon dialog behavior" on page 597 and "Connection String keywords" on page 586 for more implementation issues.

Get information regarding a driver or data source	Byte	SQLGetInfo	Returns information about a specific driver and data source.	Special attributes returned differently based on attributes and keywords. The information that is returned by SQLGetInfo can vary depending on which keywords and attributes are in use. The InfoType options that are affected are: <ul style="list-style-type: none"> • SQL_CATALOG_NAME_SEPARATOR – By default a period is returned. If the connection string keyword NAM is set to 1, a comma is returned. • SQL_CURSOR_COMMIT_BEHAVIOR, SQL_CURSOR_ROLLBACK_BEHAVIOR – By default SQL_CB_PRESERVE is returned. If the connection attribute, 1204, is set SQL_CB_DELETE is returned. • SQL_DATA_SOURCE_READ_ONLY – By default N is returned. If the connection string keyword CONNTYPE is set to 0 then Y is returned. • SQL_IDENTIFIER_QUOTE_CHAR – By default a double-quote mark is returned. If the application in use is MS QUERY (MSQRY32) then a single blank is returned. • SQL_IDENTIFIER_CASE – By default SQL_IC_UPPER is returned. If the connection string keyword DEBUG has the option 2 set then SQL_IC_MIXED is returned. • SQL_MAX_QUALIFIER_NAME_LEN – By default 18 is returned. If the connection string keyword DEBUG has the 8 bit set then 0 is returned.
--	------	------------	--	--

	N/A	SQLGetTypeInfo	Returns information about supported data types.	<p>Different result sets can be seen when running to different iSeries server versions. For example, the BIGINT data type is only in the result set when running to V4R5 or later servers.</p> <p>The "LONG VARCHAR" data type is not returned in the result set. This is due to problems that were seen with some applications expecting to specify a length with this type. "LONG VARCHAR FOR BIT DATA" and "LONG VARGRAPHIC" are also not returned for similar reasons.</p> <p>In the TYPE_NAME column, when a data type requires a value to be in parentheses, the parentheses are included in the data type name. However the parentheses are omitted when the parentheses would end up at the end of the data type string. In the following string example, the "CHAR" data type is followed by parenthesis while the "DATA" data type is not followed by parentheses: "CHAR() FOR BIT DATA".</p> <p>The setting for the connection string keyword GRAPHIC affects whether the driver returns graphic (DBCS) data types as supported types or not. See "ODBC data types and how they correspond to DB2 UDB database types" on page 598 for more information.</p>
Set and retrieve driver attributes	Byte	SQLSetConnectAttr	Sets a connection option.	
	Byte	SQLGetConnectAttr	Returns the value of a connection option.	
	N/A	SQLSetEnvAttr	Sets an environment option.	
	N/A	SQLGetEnvAttr	Returns the value of an environment option.	

	Byte	SQLSetStmtAttr	Sets a statement option.	<p>The SQL_ATTR_PARAMSET_SIZE, SQL_ATTR_ROW_ARRAY_SIZE, SQL_DESC_ARRAY_SIZE, and SQL_ROWSET_SIZE attributes support up to 32767 rows.</p> <p>SELECT statements that contain the FOR FETCH ONLY or FOR UPDATE clause override the current setting of SQL_ATTR_CONCURRENCY attribute. An error is not returned during the SQLExecute or SQLExecDirect if the SQL_ATTR_CONCURRENCY setting conflicts with the clause in the SQL statement.</p> <p>The following are not supported:</p> <ul style="list-style-type: none"> • SQL_ATTR_ASYNC_ENABLE • SQL_ATTR_RETRIEVE_DATA • SQL_ATTR_SIMULATE_CURSOR • SQL_ATTR_USE_BOOKMARKS • SQL_ATTR_FETCH_BOOKMARK_PTR <p>Setting SQL_ATTR_MAX_ROWS is supported, but only helps performance with static cursors. The full result set is still built with the other cursor types even if this option is set.</p>
	Byte	SQLGetStmtAttr	Returns the value of a statement option.	<p>The following are not supported:</p> <ul style="list-style-type: none"> • SQL_ATTR_ASYNC_ENABLE • SQL_ATTR_RETRIEVE_DATA • SQL_ATTR_SIMULATE_CURSOR • SQL_ATTR_USE_BOOKMARKS • SQL_ATTR_FETCH_BOOKMARK_PTR
Set and retrieve descriptor fields	Byte	SQLGetDescField	Returns a piece of information from a descriptor.	
	Char	SQLGetDescRec	Returns several pieces of information from a descriptor.	
	Byte	SQLSetDescField	Sets a descriptor field.	<p>Can not set descriptor fields for an IRD other than SQL_DESC_ARRAY_STATUS_PTR and SQL_DESC_ROWS_PROCESSED_PTR.</p> <p>Does not support named parameters.</p>
	Char	SQLSetDescRec	Sets several options for a descriptor.	
	N/A	SQLCopyDesc	Copies information from one descriptor to another descriptor.	SQLCopyDesc does not support named parameters.

<p>Prepare SQL requests</p>	Char	SQLPrepare	Prepares an SQL statement for later processing.	<p>Packages are created the first time a SQL statement is prepared for that Connection. This results in the first prepare taking slightly longer to complete than it would normally take. If there are any problems with a pre-existing package the first prepare may return an error depending on the setting for the package as specified in the DSN setup GUI. On the Package tab of the DSN setup GUI are default package settings. These settings are used when package settings have not already been customized for that application. Note, these are not global settings</p> <p>By default, the driver sends SQL statement text to the host in the EBCDIC CCSID associated with your job. To enable the driver to send SQL statement text to the host in Unicode you need to set the UNICODESQL keyword to 1. Note that when sending Unicode SQL statements the driver will generate a different package name to avoid collisions with existing packages that contain EBCDIC SQL statements. Setting the connection string keyword UNICODESQL allows an application to specify Unicode data for literals in the SQL statement.</p> <p>For information on which escape sequences and scalar functions the driver supports see "SQLPrepare / SQLNativeSQL escape sequences and scalar functions:" on page 600.</p>
	Byte	SQLBindParameter	Assigns storage for a parameter in an SQL statement. See "Parameter markers" on page 567 for additional information.	<p>Data conversions are made directly from the C type that is specified to the actual host parameter (column) data type.</p> <p>The SQL data type and column size that are specified are ignored.</p> <p>Conversions that involve character data convert directly from the client codepage to the column CCSID.</p> <p>The driver returns an error during the execution of the SQL statement if SQL_DEFAULT_PARAM is specified for the Strlen_or_IndPtr parameter.</p> <p>Default parameters are not supported by the DB2 UDB database. The driver handles the binding of a parameter with the SQL_DEFAULT_PARAM option by returning an error with an SQL State of 07S01 during the execution of the CALL statement.</p>
	Char	SQLGetCursorName	Returns the cursor name associated with a statement handle.	The driver will upper case all cursor names without double-quotes around the name.

	Char	SQLSetCursorName	Specifies a cursor name.	<p>The cursor name is converted to capital letters if it is not entered in quotes. Cursor names that are entered in quotes are not converted. For example, myCursorName becomes MYCURSORNAME while "myCursorName" is treated as myCursorName, with a length of 14 since the quotes are included in the length.</p> <p>The driver supports only these characters in cursor names: "", a-z, A-Z, 0-9, or _. No error will be returned by SQLSetCursorName if an invalid name is entered, however, an error will be returned later when trying to use an invalid name.</p> <p>The cursor name can only be 18 characters long, including the leading and trailing double quotes if they exist, and must be in characters that can be translated from UNICODE to ANSI.</p> <p>If an application wishes to use a DRDA connection through ODBC then they will have the following restrictions:</p> <ul style="list-style-type: none"> • Cursor name changes are not allowed during the DRDA connection. • Cursor names will be changed by the driver and should be checked via SQLGetCursorName after the cursor is open. (after SQLExecute or SQLExecDirect).
Submit requests	N/A	SQLExecute	Runs a prepared statement.	SQLExecute is affected by the settings of several of the connection string keywords such as PREFETCH, CONNTYPE, CMT, and LAZYCLOSE. Refer to "Connection String keywords" on page 586 for descriptions of these keywords.
	Char	SQLExecDirect	Runs a statement.	See SQLPrepare and SQLExecute.
	Char	SQLNativeSQL	Returns the text of an SQL statement as translated by the driver.	
	Char	SQLDescribeParam	Returns the description for a specific parameter in a statement.	
	N/A	SQLNumParams	Returns the number of parameters in a statement.	
	N/A	SQLParamData	Returns the storage value assigned to a parameter for which data will be sent at run time (useful for long data values).	

	Byte	SQLPutData	Send part or all of a data value for a parameter (useful for long data values).	
Retrieve results and related information	N/A	SQLRowCount	Returns the number of rows that are affected by an insert, update, or delete request.	This API has been extended to also contain the cursor row count for a result set using a static cursor to V5R1 or later server versions.
	N/A	SQLNumResultCols	Returns the number of columns in the result set.	
	Char	SQLDescribeCol	Describes a column in the result set.	
	Byte	SQLColAttribute	Describes attributes of a column in the result set.	
	Byte	SQLBindCol	Assigns storage for a result column and specifies the data type.	
	N/A	SQLExtendedFetch	Returns rows in the result set. This is a supported 2.x ODBC API. However, new applications should use SQLFetchScroll API instead.	<p>Uses the value of the statement attribute SQL_ROWSET_SIZE instead of SQL_ATTR_ROW_ARRAY_SIZE for the rowset size.</p> <p>You can only use SQLExtendedFetch in combination with SQLSetPos and SQLGetData if the row size is 1.</p> <p>SQL_FETCH_BOOKMARK is not supported.</p> <p>The result set for catalog APIs (such as SQLTables and SQLColumns) is forward only and read only. When SQLExtendedFetch is used with result sets generated by catalog APIs, no scrolling is allowed.</p>
	N/A	SQLFetch	Returns rows in the result set. Can only be used with SQL_FETCH_FIRST and SQL_FETCH_NEXT since the cursor is forward only.	
	N/A	SQLFetchScroll	Returns rows in the result set. Can be used with scrollable cursor, which means all fetch orientation are allowed (except SQL_FETCH_BOOKMARK).	Does not support the fetch orientation of SQL_FETCH_BOOKMARK because the driver does not support bookmarks.
	Byte	SQLGetData	Returns part or all of one column of one row of a result set (useful for long data values). See "SQLFetch and SQLGetData" on page 568 for additional information.	SQLGetData can only be used with single row fetches. Errors are reported by SQLGetData if the row array size is larger than one.

	N/A	SQLSetPos	Positions a cursor within a fetched block of data.	SQL_UPDATE, SQL_DELETE, SQL_ADD are unsupported options for Operations parameter. SQL_LOCK_EXCLUSIVE, SQL_LOCK_UNLOCK are unsupported options for the LockType parameter.
	N/A	SQLBulkOperations	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.	The driver does not support SQLBulkOperations.
	N/A	SQLMoreResults	Determines whether there are more result sets available and if so, initializes processing for the next result set.	
	Byte	SQLGetDiagField	Returns a piece of diagnostic information.	The SQL_DIAG_CURSOR_ROW_COUNT option is only accurate for static cursors when running to V5R1 or later server versions.
	Char	SQLGetDiagRec	Returns additional error or status information.	
Get data source system table information	Char	SQLColumnPrivileges	Returns a list of columns and associated privileges for one or more tables.	Returns an empty result set when: <ul style="list-style-type: none"> • V5R1 or earlier servers or • V5R2 servers, when option 2 of the CATALOGOPTIONS connection string keyword is not set By default, when accessing V5R2 servers, column privilege information will be returned.
	Char	SQLColumns	Returns a list of information on columns in one or more tables.	
	Char	SQLForeignKeys	Returns a list of column names that comprise foreign keys, if they exist for a specified table.	
	Char	SQLProcedureColumns	Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.	The driver does not return information about columns that make up result sets generated by procedures. The driver only returns information about the parameters to the procedures.
	Char	SQLProcedures	Returns the list of procedure names stored in a specific data source.	

	Char	SQLSpecialColumns	Retrieves information about the optimal set of columns that uniquely identifies a row in a specified table. It also retrieves information about the columns that are automatically updated when any value in the row is updated by a transaction. If called with the SQL_BEST_ROWID option, returns all indexed columns of that table.	
	Char	SQLStatistics	Retrieves statistics about a single table and the list of indexes that are associated with the table.	
	Char	SQLTables	Returns a list of schemas, tables, or table types in the data source.	See "SQLTables Description" on page 602
	Char	SQLTablePrivileges	Returns a list of tables and the privileges that are associated with each table.	Returns an empty result set when: <ul style="list-style-type: none"> • V5R1 or earlier servers or • V5R2 servers, when option 2 of the CATALOGOPTIONS connection string keyword is not set By default, when accessing V5R2 servers, tables privilege information will be returned.
	Char	SQLPrimaryKeys	Returns the list of column name or names that comprise the primary key for a table.	
Clean up a statement	N/A	SQLFreeStmt	Ends statement processing and closes the associated cursor, and discards pending results.	
	N/A	SQLCloseCursor	Closes a cursor that is open on the statement handle.	
	N/A	SQLCancel	Cancels an SQL statement.	Not all queries can be cancelled. This is recommended only for long running queries.
	N/A	SQLEndTran	Commits or rolls back a transaction.	
Terminate a connection	N/A	SQLDisconnect	Closes the connection.	
	N/A	SQLFreeHandle	Releases resources associated with handles.	

Parameter markers: Parameter markers act as place holders for values that are supplied by the program when you instruct the data source to run the SQL statement. When you use **SQLPrepare**, the statement that contains the parameter markers is passed to the data source to be prepared by the SQL

| “Optimizer” on page 613. The Optimizer builds a plan of the statement and holds it for later reference.
| Each parameter marker must be associated with a program variable (strictly, a pointer to a program
| variable), and **SQLBindParameter** is used for this purpose.

| **SQLBindParameter** is a complex function. Careful study of the relevant section in the *Microsoft ODBC
| Software Development Kit and Programmer’s Reference* ISBN 1-57231-516-4 is strongly recommended.
| For most SQL statements, using **SQLBindParameter** provides input information to the function, but with
| stored procedures it also can receive data back.

| After you have prepared the statement and bound the parameters, use **SQLExecute** to set to the data
| source the current values of the associated variables.

| **SQLFetch and SQLGetData**: **SQLGetData** provides an alternative to **SQLBindCol** to retrieve data from
| the columns of a retrieved row. It can only be called after calling fetch APIs and when the array size is 1.

| As a general rule, **SQLBindCol** is preferable to **SQLGetData**. There is less performance overhead; you
| need to run **SQLBindCol** only once rather than after every fetch. However, there are special
| considerations for using **SQLBindCol** in Visual Basic.

| Visual Basic moves character strings to different locations to conserve memory. If a string variable is
| bound to a column, the memory that is referenced by a subsequent **SQLFetch** may not place the data in
| the desired variable. It is likely that a **General Protection Fault** will result. A similar problem can occur
| with **SQLBindParameter**.

| Using strings in Visual Basic is not recommended. One way to avoid this problem is to use **byte arrays**.
| Byte arrays are of a fixed size and are not subject to movement in memory.

| Another circumvention is to employ Windows memory allocation API functions that are documented in the
| Microsoft Development Library Knowledge Base. However, this method involves some difficult
| programming that is not totally transportable between Windows 3.1 and later releases.

| Using **SQLGetData** rather than **SQLBindCol** and **SQLParamData** and **SQLPutData** in conjunction with
| **SQLBindParameter** produce software that is more in keeping with Visual Basic. However, this method
| involves some difficult programming.

Coding directly to ODBC APIs

Many PC applications make ODBC calls that allow the user to seamlessly access data on different platforms. Before you begin developing your own application with ODBC APIs, you should understand how an ODBC application connects to and exchanges information with a database server.

There are supported ODBC APIs that:

- Set up the ODBC environment
- Establish and end connections to data sources
- Execute SQL statements
- Clean up the ODBC environment

Coding directly to ODBC APIs topics:

- “Calling stored procedures” on page 578
- “Using large objects (LOBs) and DataLinks with iSeries Access for Windows ODBC” on page 569
- “Examples: Stored procedures” on page 633
- “Block insert and block fetch C example” on page 579
- “Example: Block inserts using Visual Basic” on page 580
- “Visual Basic: The compromise between Jet and ODBC APIs” on page 584

Using large objects (LOBs) and DataLinks with iSeries Access for Windows ODBC:

Large objects (LOBs):

Large object (LOB) data types allow applications to store large data objects as strings. Use LOBs with iSeries Access for Windows ODBC to store and access large text documents and multimedia data types. LOBs are available when connecting with a V4R4 or later iSeries server. V5R1 and pre-V5R1 iSeries Access ODBC drivers can retrieve LOBs of 15 MBs or less. V5R2 iSeries Access ODBC drivers can retrieve LOBs of 2 GBs or less.

LOB data types:

BLOB Binary large data objects

CLOB Single-byte large character data objects

DBCLOB

Double-byte character large data objects

To view an example that uses the BLOB data type:

See "Example: Using the BLOB data type"

For more information on LOBs:

See the **Using Large Objects** topic under the **Using the Object-Relational Capabilities** heading in the SQL Programming Concepts Information Center topic.

DataLinks:

DataLink data types allow you to store many types of data in a database. Data is stored as a uniform resource locator (URL). The URL points to an object, which might be an image file, sound file, text file, and so forth.

For more information on DataLinks:

See the **Using DataLinks** topic under the **Using the Object-Relational Capabilities** heading in the SQL Programming Concepts Information Center topic.

Example: Using the BLOB data type: The following is a partial C program that uses the BLOB data type:

```
BOOL params = TRUE; // TRUE if you want to use parameter markers
SQLINTEGER char_len = 10, blob_len = 400;
SQLCHAR szCol1[21], szCol2[400], szRecCol1[21], szRecCol2[400];
SQLINTEGER cbCol1, cbCol2;
SQLCHAR stmt[2048];

// Create a table with a CHAR field and a BLOB field
rc = SQLExecDirect(hstmt, "CREATE TABLE TABBLOB(COL1 CHAR(10), COL2 BLOB(400))", SQL_NTS);

strcpy(szCol1, "1234567890");
if (!params) // no parameter markers
{
    strcpy(szCol2, "414243444546"); // 0x41 = 'A', 0x42 = 'B', 0x43 = 'C', ...
    vsprintf(stmt, "INSERT INTO TABBLOB VALUES('%s', BLOB(x'%s'))", szCol1, szCol2);
}
else
{
    strcpy(szCol2, "ABCDEF"); // 'A' = 0x41, 'B' = 0x42, 'C' = 0x43, ...
    strcpy(stmt, "INSERT INTO TABBLOB VALUES(?,?)");
}

// Prepare the 'Insert' statement
rc = SQLPrepare(hstmt, stmt, SQL_NTS);

// Bind the parameter markers
if (params) // using parameter markers
{
    cbCol1 = char_len;
    rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
        char_len, 0, szCol1, char_len + 1, &cbCol1);
}
```

```

    cbCol2 = 6;
    rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_LONGVARBINARY,
                          blob_len, 0, szCol2, blob_len, &cbCol2);
}

// Execute the 'Insert' statement to put a row of data into the table
rc = SQLExecute(hstmt);

// Prepare and Execute a 'Select' statement
rc = SQLExecDirect(hstmt, "SELECT * FROM TABBLOB", SQL_NTS);

// Bind the columns
rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, szRecCol1, char_len + 1, &cbCol1);
rc = SQLBindCol(hstmt, 2, SQL_C_BINARY, szRecCol2, blob_len, &cbCol2);

// Fetch the first row
rc = SQLFetch(hstmt);
szRecCol2[cbCol2] = '\\0';

// At this point szRecCol1 should contain the data "1234567890"
// szRecCol2 should contain the data 0x414243444546 or "ABCDEF"

```

Accessing a database server with an ODBC application: An ODBC application needs to follow a basic set of steps in order to access a database server:

1. Connect to the data source.
2. Place the SQL statement string to be executed in a buffer. This is a text string.
3. Submit the statement in order that it can be prepared or immediately run.
 - Retrieve and process the results.
 - If there are errors, retrieve the error information from the driver.
4. End each transaction with a commit or rollback operation (if necessary).
5. Terminate the connection.

Establishing ODBC connections:

SQLAllocHandle with SQL_HANDLE_ENV as the handle type

- Allocates memory for an environment handle.
 - Identifies storage for global information:
 - Valid connection handles
 - Current active connection handles
 - Variable type HENV
- Must be called by application prior to calling any other ODBC function.
- Variable type HENV is defined by ODBC in the SQL.H header file provided by the C programming language compiler or by the ODBC Software Development Kit (SDK).

The header file contains a type definition for a far pointer:

```
typedef void far * HENV
```

- In C programming language this statement is coded:


```
SQLRETURN rc;
HENV henv;

rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
```
- In Visual Basic, this statement is coded:


```
Dim henv As Long
SQLAllocEnv(henv)
```

SQLAllocHandle with SQL_HANDLE_DBC as the handle type

- Allocates memory for an connection handle within the environment.
 - Identifies storage for information about a particular connection.
 - Variable type HDBC
 - Application can have multiple connection handles.
- Application must request a connection handle prior to connecting to the data source.
- In C, this statement is coded:

```
HDBC hdbc;

rc = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
```

- In Visual Basic, this statement is coded:

```
Dim hdbc As Long
SQLAllocConnect(henv,hdbc)
```

SQLSetEnvAttr

- Allows an application to set attributes of an environment.
- To be considered an ODBC 3.x application, you must set the SQL_ATTR_ODBC_VERSION to SQL_OV_ODBC3 prior to allocating a connection handle.
- In C, this statement is coded:

```
rc = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER) SQL_OV_ODBC3, SQL_IS_UIINTEGER);
```

SQLConnect

- Loads driver and establishes a connection.
- Connection handle references information about the connection.
- Data source is coded into application.

In C, this statement is coded:

```
SQLCHAR source[ ] = "myDSN";
SQLCHAR uid[ ] = "myUID";
SQLCHAR pwd[ ] = "myPWD";

rc = SQLConnect(hdbc, source, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
```

Note: SQL_NTS indicates that the parameter string is a null-terminated string.

SQLDriverConnect

- Alternative to **SQLConnect**
- Allows application to override data source settings.
- Displays dialog boxes (optional).

Executing ODBC functions:

SQLAllocHandle with SQL_HANDLE_STMT as the handle type

- Allocates memory for information about an SQL statement.
 - Application must request a statement handle prior to submitting SQL statements.
 - Variable type HSTMT.

In C, this statement is coded:

```
HSTMT hstmt;

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

SQLExecDirect

- Executes a preparable statement.
- Fastest way to submit an SQL string for one time execution.

- If rc is not equal to SQL_SUCCESS, the SQLGetDiagRec API can be used to find the cause of the error condition.

In C, this statement is coded:

```
SQLCHAR stmt[ ] = "CREATE TABLE NAMEID (ID INTEGER, NAME VARCHAR(50))";
```

```
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
```

- Return code
 - SQL_SUCCESS
 - SQL_SUCCESS_WITH_INFO
 - SQL_ERROR
 - SQL_INVALID_HANDLE

SQLGetDiagRec

To retrieve error information for an error on a statement:

In C, this statement is coded:

```
SQLSMALLINT i = 1, cbErrorMsg ;
SQLCHAR      szSQLState[6], szErrorMsg[SQL_MAX_MESSAGE_LENGTH];
SQLINTEGER nativeError;
```

```
rc = SQLGetDiagRec(SQL_HANDLE_STMT, hstmt, i, szSQLState, &nativeError, szErrorMsg,
                  SQL_MAX_MESSAGE_LENGTH, &cbErrorMsg);
```

- **szSQLState**
 - 5 character string
 - 00000 = success
 - 01004 = data truncated
 - 07001 = wrong number of parameters

Note: The previous items are only several of many possible SQL states.

- **fNativeError** - specific to data source
- **szErrorMsg** - Error Message text

Executing prepared statements: If an SQL statement is used more than once, it is best to have the statement prepared and then executed. When a statement is prepared, variable information can be passed as parameter markers, which are denoted by question marks (?). When the statement is executed, the parameter markers are replaced with the real variable information.

Preparing the statement is performed at the server. The SQL statements are compiled and the access plans are built. This allows the statements to be executed much more efficiently. When compared to using dynamic SQL to execute the statements, the result is much closer to static SQL. Extended Dynamic preserves prepared statements accross job sessions. This allows prepared statements with parameter markers to be executed multiple times within the job session even without Extended Dynamic ON. When the database server prepares the statements, it saves some of them in a special iSeries object called a package (*SQLPKG). This approach is called **Extended Dynamic SQL**. Packages are created automatically by the driver; an option is provided to turn off Package Support. This is covered in "The performance architecture of the iSeries Access for Windows ODBC driver" on page 604.

SQLPrepare

Prepares an SQL statement for execution:

In C, this statement is coded:

```
SQLCHAR szSQLstr[ ] = "INSERT INTO NAMEID VALUES (?,?)";
```

```
rc = SQLPrepare(hstmt, szSQLstr, SQL_NTS);
```

Note: SQL_NTS indicates that the string is null-terminated.

SQLBindParameter

Allows application to specify storage, data type, and length associated with a parameter marker in an SQL statement.

In the example, parameter 1 is found in a signed double word field called **id**. Parameter 2 is found in an unsigned character array called **name**. Since the last parameter is null, the driver expects that **name** is null-terminated as it will calculate the string's length.

In C, this statement is coded:

```
SQLCHAR szName[51];
SQLINTEGER id, parmLength = 50, lenParm1 = sizeof(SQLINTEGER) , lenParm2 = SQL_NTS ;

rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
                     sizeof(SQLINTEGER), 0, &id, sizeof(SQLINTEGER), &lenParm1);
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
                     parmLength, 0, szName, sizeof(szName), &lenParm2);
```

SQLExecute

Executes a prepared statement, using current values of parameter markers:

In C, this statement is coded:

```
id=500;
strcpy(szName, "TEST");
rc = SQLExecute(hstmt); // Insert a record with id = 500, name = "TEST"
id=600;
strcpy(szName, "ABCD");
rc = SQLExecute(hstmt); // Insert a record with id = 600, name = "ABCD"
```

SQLParamData / SQLPutData

Visual Basic does not directly support pointers or fixed-location ANSI character null-terminated strings. For this reason, it is best to use another method to bind Character and Binary parameters. One method is to convert Visual Basic String data types to/from an array of Byte data types and bind the array of Byte. This method is demonstrated in "Converting strings and arrays of byte" on page 575.

Another method, that should only be used for input parameters, is to supply the parameters at processing time. This is done using **SQLParamData** and **SQLPutData** APIs:

- They work together to supply parameters.
- **SQLParamData** moves the pointer to the next parameter.
- **SQLPutData** then supplies the data for that parameter.

```
's_parm is a character buffer to hold the parameters
's_parm(1) contains the first parameter
Static s_parm(2) As String
    s_parm(1) = "Rear Bumper"
    s_parm(2) = "ABC Auto Part Store"
Dim rc As Integer
Dim cbValue As Long
Dim s_insert As String
Dim hStmt As Long
Dim lPartID As Long

rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
If rc <> SQL_SUCCESS Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLAllocStmt failed.")

s_insert = "INSERT INTO ODBCSAMPLE VALUES(?, ?, ?)"

rc = SQLBindParameter(hStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, _
                    4, 0, lPartID, 4, ByVal 0)
If rc <> SQL_SUCCESS Then _
```

```

Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLBindParameter failed.")

'#define SQL_LEN_DATA_AT_EXEC_OFFSET (-100) the parms will be supplied at run time
  cbValue = -100

' Caller set 8th parameter to "ByVal 2" so driver will return
' 2 in the token when caller calls SQLParamData
  rc = SQLBindParameter(hStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                        4, 0, ByVal 2, 0, cbValue)
  If rc <> SQL_SUCCESS Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLBindParameter failed.")

  ' Caller set 8th parameter to "ByVal 3" so driver will return
' 3 in the token when caller calls SQLParamData the second time.
  rc = SQLBindParameter(hStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
                        4, 0, ByVal 3, 0, cbValue)
  If rc <> SQL_SUCCESS Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLBindParameter failed.")

' Prepare the insert statement once.
  rc = SQLPrepare(hStmt, s_insert, SQL_NTS)

  lPartID = 1
  rc = SQLExecute(hStmt) ' Execute multiple times if needed.

' Since parameters 2 and 3 are bound with cbValue set to -100,
' SQLExecute returns SQL_NEED_DATA

  If rc = SQL_NEED_DATA Then

' See comment at SQLBindParameter: token receives 2.
  rc = SQLParamData(hStmt, token)

  If rc <> SQL_NEED_DATA Or token <> 2 Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLParamData failed.")

' Provide data for parameter 2.
  rc = SQLPutData(hStmt, ByVal s_parm(1), Len(s_parm(1)))
  If rc <> SQL_SUCCESS Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLPutData failed.")

' See comment at SQLBindParameter: token receives 3.
  rc = SQLParamData(hStmt, token)
  If rc <> SQL_NEED_DATA Or token <> 3 Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLParamData failed.")

' Provide data for parameter 2.
  rc = SQLPutData(hStmt, ByVal s_parm(2), Len(s_parm(2)))
  If rc <> SQL_SUCCESS Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLPutData failed.")

' Call SQLParamData one more time.
' Since all data are provided, driver will execute the request.
  rc = SQLParamData(hStmt, token)
  If rc <> SQL_SUCCESS Then
Call DspSQLDiagRec(SQL_HANDLE_DBC, ghDbc, "SQLParamData failed.")
  Else
  Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "SQLExecute failed.")
  End If

```

Notes:

1. These two statements operate together to supply unbound parameter values when the statement is executed.
2. Each call to **SQLParamData** moves the internal pointer to the next parameter for **SQLPutData** to supply data to. After the last parameter is filled, **SQLParamData** must be called again for the statement to be executed.

3. If **SQLPutData** supplies data for parameter markers, the parameter must be bound. Use the **cbValue** parameter set to a variable whose value is `SQL_DATA_AT_EXEC` when the statement is executed.

Converting strings and arrays of byte: The following Visual Basic functions can assist in converting strings and arrays of byte:

```
Public Sub Byte2String(InByte() As Byte, OutString As String)
    'Convert array of byte to string
    OutString = StrConv(InByte(), vbUnicode)
End Sub

Public Function String2Byte(InString As String, OutByte() As Byte) As Boolean
    'vb byte-array / string coercion assumes Unicode string
    'so must convert String to Byte one character at a time
    'or by direct memory access

    Dim I As Integer
    Dim SizeOutByte As Integer
    Dim SizeInString As Integer

    SizeOutByte = UBound(OutByte)
    SizeInString = Len(InString)
    'Verify sizes if desired

    'Convert the string
    For I = 0 To SizeInString - 1
        OutByte(I) = AscB(Mid(InString, I + 1, 1))
    Next I
    'If size byte array > len of string pad with Nulls for szString
    If SizeOutByte > SizeInString Then 'Pad with Nulls
        For I = SizeInString To SizeOutByte - 1
            OutByte(I) = 0
        Next I
    End If

    String2Byte = True
End Function

Public Sub ViewByteArray(Data() As Byte, Title As String)
    'Display message box showing hex values of byte array

    Dim S As String
    Dim I As Integer
    On Error GoTo VBANext

    S = "Length: " & Str(UBound(Data)) & " Data (in hex):"
    For I = 0 To UBound(Data) - 1
        If (I Mod 8) = 0 Then
            S = S & " " 'add extra space every 8th byte
        End If
        S = S & Hex(Data(I)) & " "
    VBANext:
    Next I
    MsgBox S, , Title
End Sub
```

Retrieving results: Running some SQL statements returns results to the application program. Running an SQL SELECT statement returns the selected rows in a result set. The **SQLFetch** API then sequentially retrieves the selected rows from the result set into the application program's internal storage. In order to work with all of the rows in a result set, call the **SQLFetch** API until no more rows are returned.

You also may issue a **Select** statement where you do not specify what columns you want returned. For example, `SELECT * FROM RWM.DBFIL` selects all columns. You may not know what columns or how many columns will be returned.

SQLNumResultCols

Returns the number of columns in a result set.

- A storage buffer that receives the information is passed as a parameter.

```
SQLSMALLINT nResultCols;
```

```
rc = SQLNumResultCols(hstmt, &nResultCols);
```

SQLDescribeCol

Returns the result descriptor for one column in a result set.

- **Column name**
- **Column type**
- **Column size**

This is used with **SQLNumResultCols** to retrieve information about the columns returned.

Using this approach, as opposed to hard coding the information in the program, makes for more flexible programs.

The programmer first uses **SQLNumResultCols** to find out how many columns were returned in the result set by a select statement. Then a loop is set up to use **SQLDescribeCol** to retrieve information about each column.

In C, this statement is coded:

```
SQLCHAR szColName[51];
SQLSMALLINT lenColName, colSQLtype, scale, nullable;
SQLSMALLINT colNum = 1;
SQLINTEGER cbColDef;
```

```
rc = SQLDescribeCol(hstmt, colNum, szColName, sizeof(szColName),
                   &lenColName, &colSQLtype, &cbColDef, &scale, &nullable);
```

SQLBindCol

Assigns the storage and data type for a column in a result set:

- Storage buffer that receives the information.
- Length of storage buffer.
- Data type conversion.

In C, this statement is coded:

```
SQLSMALLINT colNum = 1;
SQLINTEGER cbColDef;
SQLINTEGER idNum, indPtr, strlen_or_indPtr;
SQLCHAR szIDName[51];
```

```
colNum = 1;
rc = SQLBindCol(hstmt, colNum, SQL_C_LONG, &idNum, sizeof(SQLINTEGER), &indPtr);
colNum = 2;
rc = SQLBindCol(hstmt, colNum, SQL_C_CHAR, szIDName, sizeof(szIDName), &strlen_or_indPtr);
```

Note: If you use this with Visual Basic, it is recommended that you use an array of Byte data type in place of String data types.

SQLFetch

Each time **SQLFetch** is called, the driver fetches the next row. Bound columns are stored in the locations specified. Data for unbound columns may be retrieved using **SQLGetData**.

In C, this statement is coded:

```
rc = SQLFetch(hstmt);
```

Visual Basic does not directly support pointers or fixed memory location ANSI character null-terminated strings. For this reason, it is best to use another method to bind Character and Binary parameters. One method is to convert Visual Basic String data types to/from an array of Byte data types and bind the array of Byte. Another method is to use the **SQLGetData** function instead of **SQLBindCol**.

SQLGetData

Retrieves data for unbound columns after a fetch. In this example, three columns are returned and **SQLGetData** is used to move them to the correct storage location.

In C, this statement is coded:

```
SQLCHAR szTheName[16], szCredit[2];
float iDiscount, iTax;

rc = SQLFetch(hstmt);
rc = SQLGetData(hstmt, 1, SQL_C_CHAR, szTheName, 16, &strlen_or_indPtr);
rc = SQLGetData(hstmt, 2, SQL_C_FLOAT, &iDiscount, sizeof(float), &indPtr);
rc = SQLGetData(hstmt, 3, SQL_C_CHAR, szCredit, 2, &strlen_or_indPtr);
rc = SQLGetData(hstmt, 4, SQL_C_FLOAT, &iTax, sizeof(float), &indPtr);
```

In Visual Basic, this statement is coded:

```
rc = SQLFetch(hStmt)
If rc = SQL_NO_DATA_FOUND Then
    Call DisplayWarning("No record found!")
    rc = SQLCloseCursor(hStmt)
    If rc <> SQL_SUCCESS Then
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Close cursor failed.")
    End If
Else
    ' Reset lcbBuffer for the call to SQLGetData
    lcbBuffer = 0
    'Get part ID from the fetched record
    rc = SQLGetData(hStmt, 1, SQL_C_LONG, _
1PartIDReceived, Len(1PartIDReceived), lcbBuffer)
    If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then _
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, _
"Problem getting data for PartID column")

    'Get part description from the fetched record
    rc = SQLGetData(hStmt, 2, SQL_C_CHAR, _
szDescription(0), 257, lcbBuffer)
    If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then _
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, _
"Problem getting data for PartDescription column")

    'Get part provider from the fetched record
    rc = SQLGetData(hStmt, 3, SQL_C_CHAR, _
szProvider(0), 257, lcbBuffer)
    If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then _
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, _
"Problem getting data for PartProvider column")

    Call DisplayMessage("Record found!")
    rc = SQLCloseCursor(hStmt)
    If rc <> SQL_SUCCESS Then
        Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Close cursor failed.")
    End If
```

Calling stored procedures: Use stored procedures to improve the performance and function of an ODBC application. Any iSeries program can act as a stored procedure. iSeries stored procedures support input, input/output and output parameters. They also support returning result sets, both single and multiple. The stored procedure program can return a result set by specifying a cursor to return (from an embedded SQL statement) or by specifying an array of values. See "Stored procedures" on page 632 for more information.

To call a stored procedure, complete the following steps:

1. Verify that the stored procedure has been declared by using the OS/400 SQL statement CREATE PROCEDURE.

Detail: CREATE PROCEDURE should be executed only once for the life of the stored procedure. DROP PROCEDURE can be used to delete the procedure without deleting the procedure's program. DECLARE PROCEDURE also can be used, but this method has several disadvantages. The *Database Programming* book contains additional information about DECLARE PROCEDURE. View an HTML online version of the book, or print a PDF version, from the DB2 Universal Database for iSeries books online topic in the **iSeries Information Center**.

2. Prepare the call of the stored procedure by using **SQLPrepare**.
3. Bind the parameters for input and output parameters.
4. Execute the call to the stored procedure.
5. Retrieve the result set (if one is returned)

In this C example, a COBOL program named NEWORD which resided in the default iSeries library, is called. A value in a field named **szCustId** is passed, and it returns a value to a field named **szName**.

```
SQLRETURN rc;
HSTMT hstmt;
SQLCHAR Query[320];
SQLCHAR szCustId[10];
SQLCHAR szName[30];
SQLINTEGER strlen_or_indPtr = SQL_NTS, strlen_or_indPtr2 = SQL_NTS;

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

// Create the stored procedure definition.
// The create procedure could be moved to the application's
// install program so that it is only executed once.
strcpy(Query, "CREATE PROCEDURE NEWORD (:CID IN CHAR(10), :NAME OUT CHAR(30) )");
strcat(Query, " (EXTERNAL NAME NEWORD LANGUAGE COBOL GENERAL WITH NULLS)");

// Create the stored procedure
rc = SQLExecDirect(hstmt, (unsigned char *)Query, SQL_NTS);

strcpy(Query, "CALL NEWORD(?,?)");

// Prepare the stored procedure call
rc = SQLPrepare(hstmt, (unsigned char *)Query, SQL_NTS);

// Bind the parameters
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
                     10, 0, szCustId, 11, &strlen_or_indPtr);

rc = SQLBindParameter(hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, SQL_VARCHAR,
                     30, 0, szName, 31, &strlen_or_indPtr2);

strcpy (szCustId, "0000012345");
// Execute the stored procedure
rc = SQLExecute(hstmt);
```


Block insert and block fetch C example: Block inserts and block fetches can be used to enhance the performance of an ODBC application. They allow you to insert or retrieve rows in blocks, rather than individually. This reduces the data flows and line turnaround between the client and the server. Block fetches can be accomplished using either the SQLFetch (forward only) or SQLExtendedFetch or SQLFetchScroll API.

A block fetch:

- Returns a block of data (one row set) in the form of an array for each bound column.
- Scrolls through the result set according to the setting of a scroll type argument; forward, backward, or by row number.
- Uses the row set size specified with the SQLSetStmtAttr API.

The C example below does a block insert of 6 rows of data followed by two block fetches of two rows.

```
#define NUM_ROWS_INSERTED 6
#define NAME_LEN          10

HSTMT hstmt;
SQLINTEGER rowcnt = NUM_ROWS_INSERTED;
SQLCHAR itemNames[NUM_ROWS_INSERTED][NAME_LEN+1] = { "puzzle   ", "candy bar ",
  "gum       ", "kite      ", "toy car   ", "crayons   " };
SQLINTEGER itemPrices[NUM_ROWS_INSERTED] = { 5, 2, 1, 10, 3, 4 };
SQLCHAR queryItemNames[NUM_ROWS_INSERTED][NAME_LEN+1]; // Name return array
SQLINTEGER queryItemPrices[NUM_ROWS_INSERTED]; // price return array
SQLINTEGER cbqueryItemNames[NUM_ROWS_INSERTED], cbqueryItemPrices[NUM_ROWS_INSERTED];

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

rc = SQLExecDirect(hstmt, "CREATE TABLE ITEMS (NAME VARCHAR(10), PRICE INT)", SQL_NTS);

// set the paramset size to 6 as we are block inserting 6 rows of data
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)rowcnt, SQL_IS_INTEGER);

// bind the arrays to the parameters
rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_VARCHAR,
  NAME_LEN, 0, itemNames[0], NAME_LEN + 1, NULL);
rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER,
  NUM_ROWS_INSERTED, 0, &itemPrices[0],
  sizeof(long), NULL);

// do the block insert
rc = SQLExecDirect(hstmt, "INSERT INTO ITEMS ? ROWS VALUES(?,?)", SQL_NTS);

// set up things for the block fetch

// We set the concurrency below to SQL_CONCUR_READ_ONLY, but since SQL_CONCUR_READ_ONLY
// is the default this API call is not necessary. If update was required then you would use
// SQL_CONCUR_LOCK value as the last parameter.
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_CONCURRENCY, (SQLPOINTER)SQL_CONCUR_READ_ONLY,
  SQL_IS_INTEGER);

// We set the cursor type to SQL_CURSOR_FORWARD_ONLY, but since SQL_CURSOR_FORWARD_ONLY
// is the default this API call is not necessary.
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_CURSOR_TYPE,
  (SQLPOINTER)SQL_CURSOR_FORWARD_ONLY, SQL_IS_INTEGER);

// We want to block fetch 2 rows at a time so we need to set SQL_ATTR_ROW_ARRAY_SIZE to 2.
// If we were going to use SQLExtendedFetch instead of SQLFetchScroll we would instead need
// to set the statement attribute SQL_ROWSET_SIZE to 2.
```

```

rc = SQLSetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE, (SQLPOINTER)2, SQL_IS_INTEGER);

rc = SQLExecDirect(hstmt, "SELECT NAME, PRICE FROM ITEMS WHERE PRICE < 5", SQL_NTS);

// bind arrays to hold the data for each column in the result set
rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, queryItemNames, NAME_LEN + 1, cbqueryItemNames);
rc = SQLBindCol(hstmt, 2, SQL_C_LONG, queryItemPrices, sizeof(long), cbqueryItemPrices);

// We know that there are 4 rows that fit the criteria for the SELECT statement so we call
// two fetches to get all the data
rc = SQLFetchScroll(hstmt, SQL_FETCH_FIRST, 0);
// at this point 2 rows worth of data will have been fetched and put into the buffers
// that were bound by SQLBindCol

rc = SQLFetchScroll(hstmt, SQL_FETCH_NEXT, 0);
// at this point 2 rows worth of data will have been fetched and put into the buffers
// that were bound by SQLBindCol. Note that this second fetch overwrites the data in
// those buffers with the new data
// ...
// Application processes the data in bound columns...
// ...

```

Example: Block inserts using Visual Basic: Block inserts allow you to:

- Insert blocks of records with one SQL call.
- Reduces the flows between the client and server.

See “Block insert and block fetch C example” on page 579 for additional information.

The next example is a Visual Basic block insert that is significantly faster than a “parameterized” insert.

```

| Dim cbNTS(BLOCKSIZE - 1) As Long 'NTS array
| Dim lCustnum(BLOCKSIZE - 1) As Long 'Customer number array
|
| '2nd parm passed by actual length for demo purposes
| Dim szLstNam(7, BLOCKSIZE - 1) As Byte 'NOT USING NULL ON THIS PARM
| Dim cbLenLstNam(BLOCKSIZE - 1) As Long 'Actual length of string to pass
| Dim cbMaxLenLstNam As Long 'Size of one array element
|
| 'These will be passed as sz string so size must include room for null
| Dim szInit(3, BLOCKSIZE - 1) As Byte 'Size for field length + null
| Dim szStreet(13, BLOCKSIZE - 1) As Byte 'Size for field length + null
| Dim szCity(6, BLOCKSIZE - 1) As Byte 'Size for field length + null
| Dim szState(2, BLOCKSIZE - 1) As Byte 'Size for field length + null
| Dim szZipCod(5, BLOCKSIZE - 1) As Byte 'Size for field length + null
|
| Dim fCdtLmt(BLOCKSIZE - 1) As Single
| Dim fChgCod(BLOCKSIZE - 1) As Single
| Dim fBalDue(BLOCKSIZE - 1) As Single
| Dim fCdtDue(BLOCKSIZE - 1) As Single
|
| Dim irow As Long ' row counter for block errors
| Dim lTotalRows As Long ' ***** Total rows to send *****
| Dim lNumRows As Long ' Rows to send in one block
| Dim lRowsLeft As Long ' Number of rows left to send
|
| Dim I As Long
| Dim J As Long
| Dim S As String
| Dim hStmt As Long
|
| ' This program needs QCUSTCDT table in your own collection.
| ' At the iSeries server command line type:
| '====> CRTLIB SAMPCOLL

```

```

| '====> CRTDUPOBJ OBJ(QCUSTCDT) FROMLIB(QIWS)
| '
| '         OBJTYPE(*FILE) TOLIB(SAMPCOLL) NEWOBJ(*SAME)
| '====> CHGPF FILE(SAMPCOLL/QCUSTCDT) SIZE(*NOMAX)
| '====> CLRPFM FILE(SAMPCOLL/QCUSTCDT)
|
| '***** Start *****
| S = "Number of records to insert into QCUSTCDT. "
| S = S & "Use menu option Table Mgmt, Create QCUSTCDT to "
| S = S & "create the table. Use Misc, iSeries Cmd and CLRPFM "
| S = S & "command if you wish to clear it"
| S = InputBox(S, gAppName, "500")
| If Len(S) = 0 Then Exit Sub
|
| lTotalRows = Val(S)          'Total number to insert
|
| rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
| If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert
|
| rc = SQLPrepare(hStmt, _
|     "INSERT INTO QCUSTCDT ? ROWS VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", _
|     SQL_NTS)
| If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert
|
| rc = SQLBindParameter(hStmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, SQL_INTEGER, _
|     10, 0, lCustnum(0), 0, ByVal 0)
| If (rc = SQL_ERROR) Then _
| Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
|
| 'Pass first parm w/o using a null
| cbMaxLenLstNam = UBound(szLstNam, 1) - LBound(szLstNam, 1) + 1
| rc = SQLBindParameter(hStmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
|     8, _
|     0, _
|     szLstNam(0, 0), _
|     cbMaxLenLstNam, _
|     cbLenLstNam(0))
| If (rc = SQL_ERROR) Then _
| Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
|
| rc = SQLBindParameter(hStmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
|     3, 0, szInit(0, 0), _
|     UBound(szInit, 1) - LBound(szInit, 1) + 1, _
|     cbNTS(0))
| If (rc = SQL_ERROR) Then _
| Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
|
| rc = SQLBindParameter(hStmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
|     13, 0, szStreet(0, 0), _
|     UBound(szStreet, 1) - LBound(szStreet, 1) + 1, _
|     cbNTS(0))
| If (rc = SQL_ERROR) Then _
| Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
|
| rc = SQLBindParameter(hStmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
|     6, 0, szCity(0, 0), _
|     UBound(szCity, 1) - LBound(szCity, 1) + 1, _
|     cbNTS(0))
| If (rc = SQL_ERROR) Then _
| Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
|
| rc = SQLBindParameter(hStmt, 6, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, _
|     2, 0, szState(0, 0), _
|     UBound(szState, 1) - LBound(szState, 1) + 1, _
|     cbNTS(0))
| If (rc = SQL_ERROR) Then _
| Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

```

```

rc = SQLBindParameter(hStmt, 7, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_NUMERIC, _
                    5, 0, szZipCod(0, 0), _
                    UBound(szZipCod, 1) - LBound(szZipCod, 1) + 1, _
                    cbNTS(0))
If (rc = SQL_ERROR) Then
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

rc = SQLBindParameter(hStmt, 8, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                    4, 0, fCdtLmt(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

rc = SQLBindParameter(hStmt, 9, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                    1, 0, fChgCod(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
rc = SQLBindParameter(hStmt, 10, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                    6, 2, fBalDue(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")
rc = SQLBindParameter(hStmt, 11, SQL_PARAM_INPUT, SQL_C_FLOAT, SQL_NUMERIC, _
                    6, 2, fCdtDue(0), 0, ByVal 0)
If (rc = SQL_ERROR) Then
Call DspSQLDiagRec(SQL_HANDLE_STMT, hStmt, "Problem: Bind Parameter")

lRowsLeft = lTotalRows      'Initialize row counter
For J = 0 To ((lTotalRows - 1) \ BLOCKSIZE)
    For I = 0 To BLOCKSIZE - 1
        cbNTS(I) = SQL_NTS      ' init array to NTS
        lCustnum(I) = I + (J * BLOCKSIZE) 'Customer number = row number
        S = "Nam" & Str(lCustnum(I))    'Last Name
        cbLenLstNam(I) = Len(S)
        rc = String2Byte2D(S, szLstNam(), I)
        'Debug info: Watch address to see layout
        addr = VarPtr(szLstNam(0, 0))
        'addr = CharNext(szLstNam(0, I))      'address of 1,I
        'addr = CharPrev(szLstNam(0, I), szLstNam(1, I)) 'address of 0, I
        'addr = CharNext(szLstNam(1, I))
        'addr = CharNext(szLstNam(6, I))      'should point to null (if used)
        'addr = CharNext(szLstNam(7, I))      'should also point to next row

        rc = String2Byte2D("DXD", szInit, I)
        'Vary the length of the street
        S = Mid("1234567890123", 1, ((I Mod 13) + 1))
        rc = String2Byte2D(S, szStreet, I)

        rc = String2Byte2D("Roches", szCity, I)
        rc = String2Byte2D("MN", szState, I)
        rc = String2Byte2D("55902", szZipCod, I)
        fCdtLmt(I) = I
        fChgCod(I) = 1
        fBalDue(I) = 2 * I
        fCdtDue(I) = I / 2
    Next I

    lNumRows = lTotalRows Mod BLOCKSIZE      ' Number of rows to send in this block
    If (lRowsLeft >= BLOCKSIZE) Then _
        lNumRows = BLOCKSIZE                ' send remainder or full block

    irow = 0
    lRowsLeft = lRowsLeft - lNumRows

    rc = SQLSetStmtAttr(hStmt, SQL_ATTR_PARAMSET_SIZE, lNumRows, 0)
    If (rc = SQL_ERROR) Then GoTo errBlockInsert

    rc = SQLSetStmtAttr(hStmt, SQL_ATTR_PARAMS_PROCESSED_PTR, irow, 0)

```

```

|         If (rc = SQL_ERROR) Then GoTo errBlockInsert
|
|         rc = SQLExecute(hStmt)
|         If (rc = SQL_ERROR) Then
|             S = "Error on Row: " & Str(irow) & Chr(13) & Chr(10)
|             MsgBox S, , gAppName
|             GoTo errBlockInsert
|         End If
|     Next J
|     rc = SQLEndTran(SQL_HANDLE_DBC, ghDbc, SQL_COMMIT)
|     If (Not (rc = SQL_SUCCESS Or rc = SQL_SUCCESS_WITH_INFO)) Then GoTo errBlockInsert
|     rc = SQLFreeHandle(SQL_HANDLE_STMT, hStmt)
|     Exit Sub
|
| errBlockInsert:
|     rc = SQLEndTran(SQL_HANDLE_DBC, ghDbc, SQL_ROLLBACK)
|     rc = SQLFreeHandle(SQL_HANDLE_STMT, hStmt)
|
| Public Function String2Byte2D(InString As String, OutByte() As Byte, RowIdx As Long)
| As Boolean
|     'VB byte arrays are layed out in memory opposite of C. The string would
|     'be by column instead of by row so must flip flop the string.
|     'ASSUMPTIONS:
|     '   Byte array is sized before being passed
|     '   Byte array is padded with nulls if > size of string
|
|     Dim I As Integer
|     Dim SizeOutByte As Integer
|     Dim SizeInString As Integer
|
|     SizeInString = Len(InString)
|     SizeOutByte = UBound(OutByte, 1)
|
|     'Convert the string
|     For I = 0 To SizeInString - 1
|         OutByte(I, RowIdx) = AscB(Mid(InString, I + 1, 1))
|     Next I
|     'If byte array > len of string pad
|     If SizeOutByte > SizeInString Then
|         'Pad with Nulls
|         For I = SizeInString To SizeOutByte - 1
|             OutByte(I, RowIdx) = 0
|         Next I
|     End If
|     'ViewByteArray OutByte, "String2Byte"
|     String2Byte2D = True
| End Function

```

Ending ODBC functions: The last procedure that must be completed before ending an ODBC application is to free the resources and memory allocated by the application. This must be done so that they are available when the application is run the next time.

SQLFreeStmt

Stops processing associated with a specific statement handle.

```
rc = SQLFreeStmt(hstmt, option); // option can be SQL_CLOSE, SQL_RESET_PARAMS. or SQL_UNBIND
```

SQL_CLOSE

Closes the cursor associated with the statement handle, and discards all pending results. Alternately, you can use `SQLCloseCursor`.

SQL_RESET_PARAMS

Releases all common buffers that are bound by `SQLBindParameter`.

SQL_UNBIND

Releases all common buffers that are bound by `SQLBindCol`.

SQLFreeHandle with SQL_HANDLE_STMT as the handle type

Frees all resources for this statement.

```
rc = SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
```

SQLDisconnect

Closes the connection associated with a specific connection handle.

```
rc = SQLDisconnect(hdbc);
```

SQLFreeHandle with SQL_HANDLE_DBC as the handle type

Releases connection handle and frees all memory associated with a connection handle.

```
rc = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

SQLFreeHandle with SQL_HANDLE_ENV as the handle type

Frees environment handle and releases all memory associated with the environment handle.

```
rc = SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

Visual Basic: The compromise between Jet and ODBC APIs: While the database objects are easy to code, they sometimes can adversely affect performance. Coding to the APIs and to stored procedures can be a frustrating endeavor.

Fortunately, if you are using Visual Basic Enterprise Edition in the Windows 95 environment, there are additional options. These options are a good compromise between the usability of database objects and the high performance of APIs: Remote Data Objects (RDO) and Remote Data Control (RDC).

RDO is a thin layer over the ODBC APIs. It provides a simple interface to advanced ODBC functionality without requiring programming to the API level. It does not have all of the overhead of the Jet Engine controlled Data Access Object (DAO) or its SQL optimizer. Yet it maintains a nearly identical programming interface as the DAOs. If you understand programming to the DAO, then switching over to the RDO is relatively simple compared to trying to switch over to API calls.

The following are differences between DAO and RDO:

- The DAO model is used for ISAM, Access and ODBC databases. The RDO model is designed for ODBC databases only, and it has been optimized for Microsoft SQL Server 6.0 and Oracle.
- The RDO model can have better performance, with the processing being done by the server and not the local machine. Some processing is done locally with the DAO model, so performance may not be as good.
- The DAO model uses the Jet Engine. The RDO model does not use Jet Engine, it uses the ODBC backend engine.
- The RDO model has the capability to perform synchronous or asynchronous queries. The DAO model has limitations in performing these type of queries.
- The RDO model can perform complex cursors, which are limited in the DAO model.

The RDC is a data control similar to the standard data control. This means that where ever you might have used a data control, and the Jet engine, you now can use the RDC. You can drag a "data aware" control on your form. It can be bound to an RDC, as it could be bound to a regular data control.

Some of the advanced ODBC functionality the RDO allows is prepared SQL statements, multiple result sets, and stored procedures. When Jet executes a SQL statement dynamically it is a two-step process on the iSeries server. In the first step, the iSeries server looks at the statement and determines the best plan to retrieve the data requested based on the current database schema. In the second step, that plan is used to actually retrieve the data. Creating that plan can be expensive in terms of time because the iSeries server has to evaluate many alternatives and determine the best way to access the data. There is an alternative to forcing the iSeries server to re-create the access plan every time a SQL statement is run. The **CreatePreparedStatement** method of the **rdoConnection** object allows you to compile a data access plan on the iSeries server for an SQL statement without executing it. You can even include parameters in prepared statements, so you can pass new selection criteria every time you run the select statement.

The following sample Visual Basic code will show how to prepare a SQL statement with a parameter marker and run it multiple times with different values.

Visual Basic 4.0 RDO sample code

```
Private Sub Command1_Click()

    Dim rdoEnv As rdoEnvironment
    Dim rdoConn As rdoConnection
    Dim rdoPS As rdoPreparedStatement
    Dim rdoRS As rdoResultset
    Dim strSQL As String

    A → strSQL = "Select * from Customer where CUSTNUM=?"
    Set rdoEnv = rdoCreateEnvironment("TestEnv", "GUEST", "GUEST")
    Set rdoConn = rdoEnv.OpenConnection("Customer Data", rdDriverComplete)
    Set rdoPS = rdoConn.CreatePreparedStatement("MyFirstPS", strSQL)

    B → rdoPS.rdoParameters(0).Value = "17"
    Set rdoRS = rdoPS.OpenResultset()
    Debug.Print rdoRS("CUSTNAME"), rdoRS.RowCount

    C → rdoRS.MoreResults

    rdoPS.rdoParameters(0).Value = "13"
    rdoRS.Requery
    Debug.Print rdoRS("CUSTNAME"), rdoRS.RowCount

    Debug.Print "Done"

End Sub
```

Label A shows where the SQL statement is defined. Notice that the statement does not include a specific for the CUSTNUM, but has a question mark for the value. The question mark signifies that this value is a parameter of the prepared statement. Before you can create a result set with the prepared statement, you must set the value of any parameters in the statement.

Label B shows where the value for the parameter is defined. Notice that the first parameter is defined as 0 not as 1. Once the value for the parameter is set you can run the **OpenResultSet** method of the **rdoPreparedStatement** to return the requested data.

Before you can requery a prepared statement on the iSeries server, you have to make sure that the cursor has been completely processed and closed. Label C shows the **MoreResults** method of the **rdoResultSet** being used to do this. The **MoreResults** method queries the database. It determines if there is any more data in the result set to be processed, or if the result set has been processed completely. Once the cursor has been fully processed you can reset the parameter value and run the **ReQuery** method of the **rdoResultSet** to open a new result set.

ODBC API return codes

Every ODBC API function returns a value of type SQLRETURN (a short integer). There are seven possible return codes, and associated with each is a manifest constant. The following list provides an explanation of each particular code. Some return codes can be interpreted as an error on the function call. Others indicate success. Still others indicate that more information is needed or pending.

A particular function may not return all possible codes. See the *Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference, Version 3.0 ISBN 1-57231-516-4*. for possible values, and for the precise interpretation for that function.

Pay close attention to return codes in your program, particularly those that are associated with the processing of SQL statements processing and with data source data access. In many instances the return code is the only reliable way of determining the success of a function.

SQL_SUCCESS

Function has completed successfully; no additional information available.

SQL_SUCCESS_WITH_INFO

Function completed successfully; possibly with a nonfatal error. The application can call SQLGetDiagRec to retrieve additional information.

SQL_NO_DATA_FOUND

All rows from the result set have been fetched.

SQL_ERROR

Function failed. The application can call SQLGetDiagRec to retrieve error information.

SQL_INVALID_HANDLE

Function failed due to an unusable environment, connection, or statement handle. Programming error.

SQL_NEED_DATA

The driver is asking the application to send parameter data values.

Implementation issues of ODBC APIs

See “ODBC 3.x APIs” on page 558 for a table of individual APIs and their associated considerations. For some global API considerations, see “ODBC API restrictions and unsupported functions” on page 597.

Related topic:

- “Connection String keywords”
- “Version and release changes in the ODBC driver behavior” on page 596
- “Signon dialog behavior” on page 597
- “ODBC data types and how they correspond to DB2 UDB database types” on page 598
- “Special connection and statement attributes” on page 599
- “SQLPrepare / SQLNativeSQL escape sequences and scalar functions:” on page 600
- “Cursor and Rowset size” on page 601
- “iSeries system name formats for ODBC Connection APIs” on page 12

Connection String keywords

The iSeries Access ODBC driver has many connection string keywords that can be used to change the behavior of the ODBC connection. These same keywords and their values are also stored when an ODBC data source is setup. When an ODBC application makes a connection, any keywords specified in the connection string override the values specified in the ODBC data source.

The following table lists connection string keywords that are recognized by the iSeries Access ODBC driver:

Table 2. iSeries Access ODBC connection string keywords

Keyword	Description	Choices	Default
General properties			

Table 2. iSeries Access ODBC connection string keywords (continued)

DSN	Specifies the name of the ODBC data source that you want to use for the connection.	Data source (DSN) name	none
DRIVER	Specifies the name of the ODBC driver that you want to use. Note: This should not be used if the DSN property has been specified.	"iSeries Access ODBC Driver"	none
PWD or Password	Specifies the password for connecting to the iSeries server.	iSeries password	none
SIGNON	Specifies what default user ID to use if the connection cannot be completed with the current user ID and password information.	0 = Use Windows user name 1 = Use default user ID 2 = None 3 = Use iSeries Navigator default 4 = Use Kerberos principal	3
SSL	Specifies whether a Secure Sockets Layer (SSL) connection is used to communicate with the server. SSL connections are only available when connecting to servers at V4R4 or later.	0 = Encrypt only the password 1 = Encrypt all clients/server communication	0
SYSTEM	Specifies the name of the iSeries server that you want to connect to.	iSeries server name	none
UID or UserID	Specifies the user ID for connecting to the iSeries server.	iSeries user ID	none
Server properties			
CMT or CommitMode	Specifies the default transaction isolation level.	0 = Commit immediate (*NONE) 1 = Read committed (*CS) 2 = Read uncommitted (*CHG) 3 = Repeatable read (*ALL) 4 = Serializable (*RR)	2
CONNTYPE or ConnectionType	Specifies the level of database access for the connection.	0 = Read/Write (all SQL statements allowed) 1 = Read/Call (SELECT and CALL statements allowed) 2 = Read-only (SELECT statements only)	0

Table 2. iSeries Access ODBC connection string keywords (continued)

DATABASE	<p>Specifies the iSeries relational database (RDB) name to connect. Note, this option is only valid to V5R2 iSeries servers. This option will be ignored when connecting to earlier pre-V5R2 servers.</p> <p>Special values for this option include specifying an empty-string or *SYSBAS. An empty-string indicates to use the user-profile's default setting for database. Specifying *SYSBAS will connect a user to the SYSBAS database (RDB name).</p>	iSeries relational database name	empty-string
DBQ or DefaultLibraries	<p>Specifies the iSeries libraries to add to the server job's library list. The libraries are delimited by commas or spaces, and "**USRLIBL" may be used as a place holder for the server job's current library list. The library list is used for resolving unqualified stored procedure calls and finding libraries in catalog API calls. IF "**USRLIBL" is not specified, the specified libraries will replace the server job's current library list.</p> <p>Note: The first library listed in this property will also be the default library, which is used to resolve unqualified names in SQL statements. To specify no default library, a comma should be entered before any libraries.</p>	iSeries libraries	"QGPL"
NAM or Naming	Specifies the naming convention used when referring to tables.	<p>0 = "sql" (as in <i>schema.table</i>)</p> <p>1 = "system" (as in <i>schema/table</i>)</p>	0
UNICODESQL	Specifies whether or not to send Unicode SQL statements to the server. IF set to 0, the driver will send EBCDIC SQL statements to the server. This option is only available when connecting to servers at V5R1 or later.	<p>0 = Send EBCDIC SQL statements to the server</p> <p>1 = Send Unicode SQL statements to the server</p>	0
Format properties			
DFT or DateFormat	Specifies the date format used in date literals within SQL statements.	<p>0 = yy/dd (*JUL)</p> <p>1 = mm/dd/yy (*MDY)</p> <p>2 = dd/mm/yy (*DMY)</p> <p>3 = yy/mm/dd (*YMD)</p> <p>4 = mm/dd/yyyy (*USA)</p> <p>5 = yyyy-mm-dd (*ISO)</p> <p>6 = dd.mm.yyyy (*EUR)</p> <p>7 = yyyy-mm-dd (*JIS)</p>	5

Table 2. iSeries Access ODBC connection string keywords (continued)

DSP or DateSeparator	Specifies the date separator used in date literals within SQL statements. This property has no effect unless the DateFormat property is set to 0 (*JUL), 1 (*MDY), 2 (*DMY), or 3 (*YMD).	<ul style="list-style-type: none"> 0 = "/" (forward slash) 1 = "-" (dash) 2 = "." (period) 3 = "," (comma) 4 = " " (blank) 	1
DEC or Decimal	Specifies the decimal separator used in numeric literals within SQL statements.	<ul style="list-style-type: none"> 0 = "." (period) 1 = "," (comma) 	0
TFT or TimeFormat	Specifies the time format used in time literals within SQL statements.	<ul style="list-style-type: none"> 0 = hh:mm:ss (*HMS) 1 = hh:mm AM/PM (*USA) 2 = hh.mm.ss (*ISO) 3 = hh.mm.ss (*EUR) 4 = hh:mm:ss (*JIS) 	0
TSP or TimeSeparator	Specifies the time separator used in time literals within SQL statements. This property has no effect unless the "time format" property is set to "hms".	<ul style="list-style-type: none"> • 0 = ":" (colon) • 1 = "." (period) • 2 = "," (comma) • 3 = " " (blank) 	0
Package properties			
DFTPGLIB or DefaultPkgLibrary	Specifies the library for the SQL package. This property has no effect unless the XDYNAMIC property is set to 1.	Library for SQL package	"QGPL"
PKG or DefaultPackage	<p>Specifies how the extended dynamic (package) support will behave. The string for this property must be in the following format: "A/DEFAULT(IBM),x,0,y,z,0"</p> <p>The x, y, and z are special attributes that need to be replaced with how the package is to be used.</p> <ul style="list-style-type: none"> • x = Specifies whether or not to add statements to an existing SQL package. • y = Specifies the action to take when SQL package errors occur. When a SQL package error occurs, the driver will return a return code based on the value of this property. • z = Specifies whether or not to cache SQL packages in memory. Caching SQL packages locally reduces the amount of communication to the server in some cases. <p>Note: This property has no effect unless the XDYNAMIC property is set to 1.</p>	<p>"A/DEFAULT(IBM),x,0,y,z,0"</p> <p>Values for x option:</p> <ul style="list-style-type: none"> • 1 = Use (Use the package, but do not put any more SQL statements into the package) • 2 = Use/Add (Use the package and add new SQL statements into the package) 	"A/DEFAULT(IBM),2,0,1,0"

Table 2. iSeries Access ODBC connection string keywords (continued)

<p>XDYDAMIC orExtendedDynamic</p>	<p>Specifies whether to use extended dynamic (package) support.</p> <p>Extended dynamic support provides a mechanism for caching dynamic SQL statements on the server. The first time a particular SQL statement is run, it is stored in a SQL package on the server. On subsequent runs of the same SQL statement, the server can skip a significant part of the processing by using information stored in the SQL package.</p> <p>Note: For more information see “Using Extended Dynamic SQL” on page 605.</p>	<p>0 = Disable extended dynamic support</p> <p>1 = Enable extended dynamic support</p>	<p>1</p>
<p>Performance properties</p>			
<p>BLOCKFETCH</p>	<p>Specifies whether or not internal blocking will be done on fetches of 1 row. When set, the driver will try to optimize the fetching of records when one record is requested by the application. Multiple records will be retrieved and stored by the driver for later retrieval by the application. When an application requests another row, the driver will not need to send another flow to the host database to get it. If not set, blocking will be used according to the application’s ODBC settings for that particular statement.</p> <p>Note: For more information on setting this option see Fine-tuning record blocking.</p>	<p>0 = Use ODBC settings for blocking</p> <p>1 = Use blocking with a fetch of 1 row</p>	<p>1</p>
<p>BLOCKSIZE or BlockSizeKB</p>	<p>Specifies the block size (in kilobytes) to retrieve from the iSeries server and cache on the client. This property has no effect unless the BLOCKFETCH property is 1. Larger block sizes reduce the frequency of communication to the server, and therefore may increase performance.</p>	<p>1</p> <p>2</p> <p>4</p> <p>8</p> <p>16</p> <p>32</p> <p>64</p> <p>128</p> <p>256</p> <p>512</p>	<p>32</p>
<p>COMPRESSION or AllowDataCompression</p>	<p>Specifies whether to compress data sent to and from the server. In most cases, data compression improves performance due to less data being transmitted between the driver and the server.</p>	<p>0 = Disable compression</p> <p>1 = Enable compression</p>	<p>1</p>

Table 2. iSeries Access ODBC connection string keywords (continued)

CONCURRENCY	<p>Specifies whether to override the ODBC concurrency setting by opening all cursors as updateable.</p> <p>Note: In the following two cases, setting this option has no effect:</p> <ol style="list-style-type: none"> 1. When building a SELECT SQL statement the FOR FETCH ONLY or FOR UPDATE clause can be added. If either of these clauses are present in a SQL statement the ODBC driver will honor the concurrency that is associated with the clause. 2. Catalog result sets are always read-only. 	<p>0 = Use ODBC concurrency settings</p> <p>1 = Open all cursors as updateable</p>	0
EXTCOLINFO or ExtendedCollInfo	<p>The extended column information affects what the SQLGetDescField and SQLColAttribute APIs return as Implementation Row Descriptor (IRD) information. The extended column information is available after the SQLPrepare API has been called. The information that is returned is:</p> <ul style="list-style-type: none"> • SQL_DESC_AUTO_UNIQUE_VALUE • SQL_DESC_BASE_COLUMN_NAME • SQL_DESC_BASE_TABLE_NAME and SQL_DESC_TABLE_NAME • SQL_DESC_LABEL • SQL_DESC_SCHEMA_NAME • SQL_DESC_SEARCHABLE • SQL_DESC_UPDATABLE <p>Note: the driver sets the SQL_DESC_AUTO_UNIQUE_VALUE flag only if a column is an identity column with the ALWAYS option over a numeric data type (such as integer). Refer to the DB2 UDB SQL Reference for details on identity columns.</p>	<p>0 = Do not retrieve extended column information</p> <p>1 = Retrieve extended column information</p>	0
LAZYCLOSE	<p>Specifies whether to delay closing cursors until subsequent requests. This will increase overall performance by reducing the total number of requests.</p> <p>Note: This option can cause problems due to the cursors still holding locks on the result set rows after the close request.</p>	<p>0 = Do not retrieve extended column information</p> <p>1 = Retrieve extended column information</p>	0

Table 2. iSeries Access ODBC connection string keywords (continued)

MAXFIELDLEN or MaxFieldLength	Specifies the maximum LOB (large object) size (in kilobytes) that can be retrieved as part of a result set. LOBs that are larger than this threshold will be retrieved in pieces using extra communication to the server. Larger LOB thresholds will reduce the frequency of communication to the server, but will download more LOB data, even if it is not used. Smaller LOB thresholds may increase frequency of communication to the server, but they will only download LOB data as it is needed. Note: Setting this property to 0 will force locators to always be used.	0 — 2097152	15360
PREFETCH	Specifies whether to prefetch data upon executing a SELECT statement. This will increase performance when accessing the initial rows in the ResultSet.	0 = Do not prefetch data 1 = Prefetch data	0
QUERYTIMEOUT	Specifies whether the driver will disable support for the query timeout attribute, SQL_ATTR_QUERY_TIMEOUT. If disabled, SQL queries will run until they finish.	0 = Disable support for the query timeout attribute 1 = Allow the query timeout attribute to be set	1
Sort properties			
LANGUAGEID	Specifies a 3-character language id to use for selection of a sort sequence. This property has no effect unless the SORTTYPE property is set to 2.	"AFR", "ARA", "BEL", "BGR", "CAT", "CHS", "CHT", "CSY", "DAN", "DES", "DEU", "ELL", "ENA", "ENB", "ENG", "ENP", "ENU", "ESP", "EST", "FAR", "FIN", "FRA", "FRB", "FRC", "FRS", "GAE", "HEB", "HRV", "HUN", "ISL", "ITA", "ITS", "JPN", "KOR", "LAO", "LVA", "LTU", "MKD", "NLB", "NLD", "NON", "NOR", "PLK", "PTB", "PTG", "RMS", "ROM", "RUS", "SKY", "SLO", "SQL", "SRB", "SRL", "SVE", "THA", "TRK", "UKR", "URD", "VIE"	"ENU"
SORTTABLE	Specifies the library and file name of a sort sequence table stored on the iSeries server. This property has no effect unless the SORTTYPE property is set to 3.	Qualified sort table name	none
SORTTYPE or SortSequence	Specifies how the server sorts records before sending them to the client.	0 = Sort based on hexadecimal values 1 = Sort based on the setting for the server job 2 = Sort based on the language set in LANGUAGEID property 3 = Sort based on the sort sequence table set in the SORTTABLE property	0

Table 2. iSeries Access ODBC connection string keywords (continued)

SORTWEIGHT	Specifies how the server treats case while sorting records. This property has no effect unless the SORTTYPE property is set to 2.	0 = Shared-Weight (uppercase and lowercase characters sort as the same character) 1 = Unique-Weight (uppercase and lowercase characters sort as different characters)	0
Catalog properties			
CATALOGOPTIONS	Specifies one or more options to affect how catalog APIs return information. To specify multiple catalog options, add the values associated with the options that you want.	To determine the value for this keyword, add the values below that are associated with each option that you want. 1 = Return information about aliases in the SQLColumns result set. 2 = Return result set information for SQLTablePrivileges and SQLColumnPrivileges. Note, this will only work with V5R2 hosts. On older hosts the driver will return an empty result set.	3
LIBVIEW or LibraryView	Specifies the set of libraries to be searched when returning information when using wildcards with catalog APIs. In most cases, use the default library list or default library option as searching all the libraries on the server will take a long time.	0 = Use default library list 1 = All libraries on the server 2 = Use default library only	0
REMARKS or ODBCRemarks	Specifies the source of the text for REMARKS columns in catalog API result sets.	0 = OS/400 object description 1 = SQL object comment	0
SEARCHPATTERN	Specifies whether the driver will interpret string search patterns and underscores in the library and table names as wildcards (search patterns). By default, % is treated as an 'any number of characters' wildcard, and _ is treated as a 'single character' wildcard.	0 = Do not treat search patterns as wildcards 1 = Treat search patterns as wildcards	1
Translation properties			
ALLOWUNSHAR or AllowUnsupportedChars	Specifies whether or not to suppress error messages which occur when characters that can not be translated (because they are unsupported) are detected.	0 = Report error messages when characters can not be translated 1 = Suppress error messages when characters can not be translated	0
CCSID	Specifies a codepage to override the default client codepage setting with.	Client codepage setting or 0 (use default client codepage setting)	0

Table 2. iSeries Access ODBC connection string keywords (continued)

GRAPHIC	This property affects the handling of the graphic (DBCS) data types of GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, and DBCLOB that have a CCSID other than Unicode (13488). This property affects two different behaviors: 1. Whether graphic fields have their lengths reported as a character count or byte count through the SQLDescribeCol API and SQLColAttribute API with the SQL_COLUMN_LENGTH option. 2. Whether graphic fields are reported as a supported type in the SQLGetTypeInfo result set	0 = Report character count, report as not supported 1 = Report character count, report as supported 2 = Report byte count, report as not supported 3 = Report byte count, report as supported	0
TRANSLATE or ForceTranslation	Specifies whether or not to convert binary data (CCSID 65535) is converted to text. Setting this property to 1 makes binary fields look like character fields.	0 = Do not convert binary data to text 1 = Convert binary data to text	0
XLATEDLL or TranslationDLL	Specifies the full path name of the DLL to be used by the ODBC driver to translate the data that is passed between the ODBC driver and the server. The DLL is loaded when a connection is established.	Full path name of the translation DLL	none
XLATEOPT or TranslationOption	Specifies a 32-bit integer translation option that is passed to the translation DLL. This parameter is optional. The meaning of this option depends on the translation DLL that is being used. Refer to the documentation provided with the translation DLL for more information. This option is not used unless the XLATEDLL property is set.	32-bit integer translation option	0
Diagnostic properties			
MAXTRACESIZE	Specifies the maximum trace size (in MB) of the internal driver trace. Specifying a value of 0 means no limit. This property has no effect unless the TRACE property has option 1 set.	0 (no limit) - 1000	0
MULTTRACEFILES or MultipleTraceFiles	Specifies whether or not trace data from the internal driver trace will be put into multiple files. A new file will be created for each thread that the application is using. This property has no effect unless the TRACE property has option 1 set.	0 = Trace data into a single file 1 = Trace data into multiple files	1

Table 2. iSeries Access ODBC connection string keywords (continued)

QAQQINILIB or QAQQINILibrary	Specifies a query options file library. When a query options file library is specified the driver will issue the command CHGQRYA passing the library name for the QRYOPLIB parameter. The command is issued immediately after the connection is established. This option should only be used when debugging problems or when recommended by support as enabling it will adversely affect performance.	Query options file library	none
SQDIAGCODE	Specifies DB2 UDB SQL diagnostic options to be set. Use only as directed by your technical support provider.	DB2 UDB SQL diagnostic options	none
TRACE	Specifies one or more trace options. To specify multiple trace options add together the values for the options that you want. For example, if you want the Database Monitor and Start Debug command to be activated on the server then the value you would want to specify is 6. These options should only be used when debugging problems or when recommended by support as they will adversely affect performance.	To determine the value for this keyword, add the values below that are associated with each option that you want. 0 = No tracing 1 = Enable internal driver tracing 2 = Enable Database Monitor 4 = Enable the Start Debug (STRDBG) command 8 = Print job log at disconnect 16 = Enable job trace	0
TRACEFILENAME	Specifies the full path name to either the file or the directory in which to put the internal driver trace data into. A path name to the file should be specified if MULTTRACEFILES is set to 0. A path name to a directory should be specified if MULTTRACEFILES is set to 1. This property has no effect unless the TRACE property has option 1 set.	Full path name to file or directory	none
Other properties			
ALLOWPROCCALLS	Specifies whether stored procedures can be called when the connection attribute, SQL_ATTR_ACCESS_MODE, is set to SQL_MODE_READ_ONLY.	0 = Do not allow stored procedures to be called 1 = Allow stored procedures to be called	0
DB2SQLSTATES	Specifies whether or not to return ODBC-defined SQL States or DB2 SQL States. Refer to the DB2 UDB SQL Reference for more details on the DB2 SQL States. This option should be used only if you have the ability to change the ODBC application's source code. If not, you should leave this option set to 0 as most applications are coded only to handle the ODBC-defined SQL States.	0 = Return ODBC-defined SQLStates 1 = Return DB2 SQL States	0

Table 2. iSeries Access ODBC connection string keywords (continued)

DEBUG	Specifies one or more debug options. To specify multiple debug options add together the values for the options that you want. In most cases you will not need to set this option.	<p>To determine the value for this keyword, add the values below that are associated with each option that you want.</p> <p>2 = Return SQL_IC_MIXED for the SQL_IDENTIFIER_CASE option of SQLGetInfo</p> <p>4 = Store all SELECT statements in the package</p> <p>8 = Return zero for the SQL_MAX_QUALIFIER_NAME_LEN option of option of SQLGetInfo</p> <p>16 = Add positioned UPDATES / DELETES into packages</p> <p>32 = Convert static cursors to dynamic cursors</p> <p>64 = Send the entire column size worth of data for variable length fields (VARCHAR, VARGRAPHIC, BLOB, etc.) Note, set this option with caution as this can have an adverse impact on performance.</p>	0
TRUEAUTOCOMMIT	Specifies whether or not to enable a true autocommit. True autocommit means that autocommit is on and is running under a isolation level other than *NONE. By default, the driver handles autocommit by running under the server isolation level of *NONE.	<p>0 = Do not use true autocommit</p> <p>1 = Use true autocommit</p>	0

Version and release changes in the ODBC driver behavior

The following list describes some of the important changes for V5R2:

- There are several new features available when using the ODBC driver to access data on a V5R2 iSeries server. These features include:
 - Ability to send Structured Query Language (SQL) statements that are 64K bytes long to the DB2 UDB database (the previous limit was 32K bytes)
 - Ability to make use of the DB2 UDB database type of ROWID
 - Ability to get back additional descriptor information, such as the base table name for a result set column
 - Ability to access multiple databases on the same iSeries server
 - Ability to retrieve meaningful information from the SQLTablePrivileges and SQLColumnPrivileges APIs
 - Ability to use Kerberos support for authenticating a user to an iSeries server
 - Ability to retrieve, regardless of the iSeries server version, more information in the result sets for the catalog APIs. The driver now queries the iSeries catalog tables directly to provide the result set for the catalog APIs.

The following list describes some of the important changes for V5R1:

- Character data for parameter markers is converted from the iSeries Access(PC) codepage directly to the column CCSID. If a new iSeries Access codepage setting was specified on the Advanced Translation Options dialog of the DSN setup GUI, it will be the iSeries Access(PC) codepage. The V4R5 driver first converted character data from the iSeries Access(PC) codepage to the job CCSID before it was converted to the column CCSID.
- Character column data is converted directly from the column CCSID to the iSeries Access(PC) codepage. If the C type specified is SQL_C_WCHAR, then the data is converted to Unicode.
- If the value type specified in SQLBindParameter is SQL_C_WCHAR, then the driver converts the parameter marker data from Unicode to the column CCSID.
- When calling SQLBindParameter for a SQL_C_CHAR to INTEGER conversion, if BufferLength is 0 and the buffer contains an empty string then an error is returned. The V4R5 driver would accept the empty string and insert the value of 0 into the table.
- The lazy close option default is 0 (OFF), and in V4R5 its default was 1 (ON).
- The prefetch option default is 0 (OFF), and in V4R5 its default was 1 (ON).
- Unicode SQL statements can be sent to V5R1 or later iSeries servers. The package names are generated differently than in V4R5 when sending Unicode SQL statements.
- Managed DSNs (created through V4R5 or earlier iSeries Navigator) are not supported. They are instead treated like a User DSN, meaning that the DSN information is not updated from the server copy.
- BIGINT data type is supported to V4R5 (or later) hosts.
- Static cursor supported to V5R1 or later hosts. In earlier hosts, and in previous iSeries Access for Windows ODBC drivers, static cursor type is mapped to dynamic.

ODBC API restrictions and unsupported functions

The way in which some functions are implemented in the iSeries Access for Windows ODBC Driver does not meet the specifications in the *Microsoft ODBC Software Development Kit Programmer's Reference*. The table below describes some global restrictions and unsupported functions. See "ODBC 3.x APIs" on page 558 for a list of individual APIs and their associated considerations.

Table 3. Limitations of ODBC API functions

Function	Description
Global considerations	No asynchronous processes are supported. However, SQLCancel can be called, from a different thread (in a multi-threaded application), to cancel a long running query. Translation DLLs are only called when converting data from buffers.
SQLSetScrollOptions (2x API)	SQL_CONCUR_ROWVER, SQL_CONCUR_VALUES are unsupported options for Concurrency parameter. The SQL_SCROLL_KEYSET_DRIVEN is mapped to SQL_SCROLL_DYNAMIC by the driver.

Signon dialog behavior

The signon dialog behavior has been simplified from the behavior seen in previous iSeries Access for Windows ODBC drivers. The signon dialog behavior is based on how your data source is set up and which ODBC API (SQLConnect, SQLDriverConnect, SQLBrowseConnect) your application uses to connect.

When configuring an ODBC data source there are two options which can influence the signon dialog behavior. These are both located on the dialog you get after clicking the **Connection Options** button on the **General** tab of the DSN Setup GUI.

Note: On the DSN setup GUI there is an option which controls whether or not a dialog prompting for signon information is allowed or not. An application that calls SQLConnect in a 3-tier environment should always choose 'Never prompt for SQLConnect'. This 3-tier application also needs to make sure it specifies the userid and password when calling SQLConnect.

- In the **Default user ID** section you can specify which default user ID to use:
 - Use Windows user name
 - Use the user ID specified below
 - None
 - Use iSeries Navigator default
 - Use Kerberos principal
- In the **Signon dialog prompting** section you can specify if the signon dialog should be prompted if your application uses the SQLConnect ODBC API.

When coding your application you have total control over how the userid, password, and signon dialog prompting will behave. The userid and password that is used is figured out in the following order:

1. Userid / Password arguments specified by the application.
 - The SQLConnect API accepts userid and password arguments.
 - The SQLDriverConnect and SQLBrowseConnect APIs accept the UID, PWD, and SIGNON connection string keywords.
2. GUI setting for Default user ID

The signon dialog prompting depends on which ODBC API is used by the application to connect. SQLConnect prompts the signon dialog if needed unless the GUI setting for Signon dialog prompting says to never prompt. SQLDriverConnect prompts the signon dialog according to the value of the DriverCompletion. A setting of SQL_DRIVER_NOPROMPT will prevent any signon dialogs from being prompted. A setting of SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE or SQL_DRIVER_COMPLETE_REQUIRED will prompt the signon dialog if needed. SQLBrowseConnect prompts the signon dialog if needed.

ODBC data types and how they correspond to DB2 UDB database types

The iSeries Access for Windows ODBC Driver maps data types between ODBC types and DB2 UDB types. The following table shows how data types are mapped between the DB2 UDB database type and ODBC SQL type.

Table 4.

3.x ODBC Data Type	DB2 UDB Database Type
SQL_BIGINT	BIGINT
SQL_BINARY	CHAR FOR BIT DATA
SQL_CHAR	CHAR or GRAPHIC
SQL_DECIMAL	DECIMAL
SQL_DOUBLE	DOUBLE
SQL_FLOAT	FLOAT
SQL_INTEGER	INTEGER
SQL_LONGVARBINARY	BLOB
SQL_LONGVARCHAR	CLOB or DBCLOB
SQL_NUMERIC	NUMERIC
SQL_REAL	REAL
SQL_SMALLINT	SMALLINT
SQL_TYPE_DATE	DATE
SQL_TYPE_TIME	TIME
SQL_TYPE_TIMESTAMP	TIMESTAMP

Table 4. (continued)

SQL_VARBINARY	VARCHAR FOR BIT DATA or LONG VARCHAR FOR BIT DATA or ROWID
SQL_VARCHAR	VARCHAR or VARGRAPHIC or LONG VARCHAR or LONG VARGRAPHIC or DATALINK
SQL_WCHAR	GRAPHIC CCSID 13488
SQL_WLONGVARCHAR	DBCLOB CCSID 13488
SQL_WVARCHAR	VARGRAPHIC CCSID 13488 or LONG VARGRAPHIC CCSID 13488

Implementation notes:

- All conversions in the Microsoft ODBC Software Development Kit Programmer’s Reference Version 3.5 are supported for these ODBC SQL data types.
- Call the ODBC API SQLGetTypeInfo to learn more about each of these data types.
- The database type of VARCHAR will be changed to LONG VARCHAR by the database if the column size that is specified is larger than 255.
- The ODBC driver does not support any of the interval SQL data types.
- 2.x ODBC applications use the SQL_DATE, SQL_TIME, and SQL_TIMESTAMP defines in place of the SQL_TYPE_DATE, SQL_TYPE_TIME, and SQL_TYPE_TIMESTAMP defines.
- LOBs (BLOB, CLOB, and DBCLOB) up to 2 GB in size are supported by V5R2 DB2 UDB databases only. Earlier releases support up to 15 MB. For more information on LOBs and datalinks see “Using large objects (LOBs) and DataLinks with iSeries Access for Windows ODBC” on page 569.

Special connection and statement attributes

The following two tables describe special connection and statement attributes supported by the iSeries Access ODBC driver. The tables include the information needed to call the SQLGetConnectAttr, SQLSetConnectAttr, SQLGetStmtAttr, and SQLSetStmtAttr APIs. Note that some of the attributes can not be both set and retrieved as indicated in the Get/Set column.

Table 5. Special connection attributes

Attribute	Get/Set	Description
1204	both	An unsigned value that controls the cursor commit behavior and cursor rollback behavior. Possible values: 0 - SQL_CB_DELETE is returned for SQLGetInfo’s SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR options. 1 - (default) SQL_CB_PRESERVE is returned for SQLGetInfo’s SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR options.
2100	both	Can be used as an alternative to using the PKG connection string keyword. This is a character string that specifies the default package library to be used. This should be set prior to preparing a statement on this connection.

Table 5. Special connection attributes (continued)

2101	both	Can be used as an alternative to using the PKG connection string keyword. This is a character string that specifies the package name to be used. This should be set prior to preparing a statement on this connection.
2103	get	Returns an unsigned integer value which is the server CCSID value (job CCSID) that the ODBC connection is dealing with. By default, SQL statements will be sent to the host in this CCSID
2104	both	Can be used as an alternative to the Divide by zero option of the DEBUG connection string keyword. This is an unsigned value indicating whether or not dividing a value by zero should return an error for data in a particular cell in the result set. Possible values: 0 - (default) A cell in a result set that contains a value calculated by dividing by zero will be returned as an error. 1 - A cell in a result set that contains a value calculated by dividing by zero will be returned as a NULL value. No error will be returned.
2106	both	An alternative to using the COMPRESSION connection string keyword. This is an unsigned integer value. Possible values: 0 = compression off, 1 = compression on
2109	set	An unsigned value specifying whether or not to trim trailing spaces from data returned from CHAR fields. This will make CHAR fields appear like VARCHAR fields as VARCHAR fields are always trimmed of trailing spaces. Possible values: 0 - (default) - don't trim CHAR fields 1 - trim CHAR fields
2110	get	Returns a character string containing information about the prestart job that the ODBC connection is using. The information is returned as a string with the following format: 10 character job name, 10 character user, 6 character job

Table 6. Special statement attributes

Attribute	Get/Set	Description
1014	get	Returns an unsigned integer value indicating how many result sets are available to be fetched. This is useful when a stored procedure has been called and an application wants to know how many result sets the stored procedure generated.
2106	both	Allows compression to be turned on an off at the statement level. possible values: 0 = compression off, 1 = compression on

SQLPrepare / SQLNativeSQL escape sequences and scalar functions:

ODBC has escape sequences that can be used to avoiding having to code directly to the syntax of a particular DBMS's version of SQL. See Microsoft's ODBC specification on how to use escape sequences. The following ODBC escape sequences are supported by the iSeries Access for Windows ODBC driver. Note, DB2 UDB supports other escape sequences than those listed below that can be used in SQL statements. Refer to the SQL programming guide for information on that.

Escape sequences:

- d
- t
- ts
- escape
- oj
- call
- ?=call – This escape sequence should be used when trying to take advantage of the DB2 UDB for iSeries support for return values from a stored procedure. The parameter marker will need to be bound as an output parameter using the SQLBindParameter API. Note, at this time stored procedures can only return values of type integer.
- fn – This escape sequence is used when using the scalar functions below. The syntax is { fn scalar_function }.

Scalar functions mapped by the ODBC driver to the DB2 UDB for iSeries SQL syntax:

- database
- hour
- insert
- length
- log
- minute
- month
- pi
- right
- second
- year

Note: To see how the driver maps the escape sequences and scalar functions to the DB2 UDB for iSeries SQL syntax the SQLNativeSQL API can be called. SQLNativeSQL allows an application to pass in an SQL statement to the ODBC driver. The ODBC driver returns an output string that is converted to the DBMS's SQL syntax.

Cursor and Rowset size

Cursor types can be set via SQLSetStmtAttr with the SQL_ATTR_CURSOR_TYPE option.

Cursor types:

- SQL_CURSOR_FORWARD_ONLY - All catalog and stored procedure result sets use this type of cursor. When a catalog or stored procedure result set has been generated the cursor type will be automatically changed to this.
- SQL_CURSOR_KEYSET_DRIVEN - mapped to SQL_CURSOR_STATIC if the host supports it, otherwise it is mapped to SQL_CURSOR_DYNAMIC
- SQL_CURSOR_DYNAMIC - supported.
- SQL_CURSOR_STATIC - A static cursor is supported to V5R1 and later iSeries servers. This cursor type is mapped to SQL_CURSOR_DYNAMIC for earlier iSeries versions.

The following factors can affect the concurrency of the cursor:

- If the SQL statement contains the "FOR UPDATE" clause the value for SQL_ATTR_CONCURRENCY will be set to SQL_CONCUR_LOCK.
- If the CONCURRENCY keyword / DSN setting is set to 1 (checked) then if the SQL statement does not have "FOR FETCH ONLY" clause in it the ODBC driver will lock records from the result set.

Rowset size:

The ODBC driver uses the value of `SQL_ROWSET_SIZE` when dealing with `SQLExtendedFetch`. The driver uses the value of `SQL_ATTR_ROW_ARRAY_SIZE` when dealing with `SQLFetch` and `SQLFetchScroll`.

When there are LOBs in a result set there is a chance that locators may be used by the driver. Locators are internal handles to LOB fields. Locators are used when the setting for the `MAXFIELDLEN` connection option has a smaller value than the size of a LOB column in the result set. Locators can improve performance in some cases as the driver only gets the data the application asks for. The downside of locators is that there is some extra communication needed with the server. When locators are not used the driver will download more LOB data even if it is not used. It is strongly encouraged that the `COMPRESSION` connection option be enabled if locators are not being used. It is recommended that locators be used only when your application retrieves part of a LOB column. In this case, using locators will avoid the retrieval of all of the LOB data. See Connection String keywords descriptions for more details on the `MAXFIELDLEN` keyword

`SQLGetData` can only be used for accessing data from single row fetches. Calling `SQLGetData` with multiple-row fetches is not supported.

Restrictions when using the 64-bit iSeries Access for Windows ODBC Driver

- MTS is not supported. For more information on MTS see Using Microsoft Transaction Server (MTS) .
- There is no SSL support. For more information on SSL see Secure Sockets Layer administration .

SQLTables Description

- The `CatalogName` parameter is ignored, with or without wildcards, since the catalog name is always the relational database name. The only time the catalog name value matters is when it must be an empty string to generate a list of libraries for the server. You must specify table names for the `TableName` parameter exactly as you would when creating a SQL statement. In other words, you must capitalize the table name unless you created the table name with double quotes around the table name. If you created the table with double quotes around the table name, you need to specify the `TableName` parameter as it appears in quotes, matching the case of the letters.
- The "OS400 library view" option on the **Catalog** tab of the DSN setup GUI only affects this API when you choose the combination that attempts to retrieve the list of libraries for that server. It does not allow you to generate a result set based on a search through multiple libraries for specific tables.
- The "Object description type" option on the **Catalog** tab of the DSN setup GUI affects the output you get in the "RESULTS" column of the result set when getting a list of tables.
- If you have a string with mixed `'_'` and `'_'` then if `SQL_ATTR_METADATA_ID` is `SQL_FALSE` then we'll treat the first `'_'` as an actual `'_'`, but the `'_'` will be treated as the wildcard. If `SQL_ATTR_METADATA_ID` is `SQL_TRUE` then the first `'_'` will be treated like an actual `'_'` and the `'_'` will also be treated like an actual `'_'`. The driver will internally convert the second `'_'` to a `'_'`.
- In order to use the wildcard character underscore (`_`) as a literal precede it with a backslash (`\`). For example, to search for only `MY_TABLE` (not `MYATABLE`, `MYBTABLE`, etc...) you need to specify the search string as `MY_TABLE`. Specifying `'\%'` in a name is invalid, as the iSeries server doesn't allow an actual `'%'` in a library or table name. When queried for the list of libraries, the driver returns the `TABLE_CAT` and `REMARKS` fields as meaningful data. The ODBC specification says to return everything, except the `TABLE_SCHEM` as nulls.

iSeries Access for Windows ODBC performance

See any of the following ODBC performance topics:

- "Performance-tuning iSeries Access for Windows ODBC" on page 603
- "Choosing an interface to access the ODBC driver" on page 639
- "Performance considerations of common end-user tools" on page 606
- "SQL performance" on page 608

- “Coding directly to ODBC APIs” on page 568
- “Visual Basic: The compromise between Jet and ODBC APIs” on page 584
- “ODBC blocked insert statement” on page 615
- “Catalog functions” on page 616
- “Exit programs” on page 617
- “Stored procedures” on page 632
- “Example: Calling CL command stored procedures” on page 638

Performance-tuning iSeries Access for Windows ODBC

A key consideration for ODBC application developers is achieving maximum **performance** from client/server applications. The following topics explore client/server performance issues in general, and address the performance implications of ODBC with popular query tools and development environments:

- “Introduction to server performance”
- “Introduction to client/server performance”
- “The performance architecture of the iSeries Access for Windows ODBC driver” on page 604

Introduction to server performance: The performance characteristics of any computing environment may be described in the following terms:

Response time

The amount of time that is required for a request to be processed

Utilization

The percentage of resources that are used when processing requests

Throughput

The volume of requests (per unit of time) that are being processed

Capacity

The maximum amount of throughput that is possible

Typically, response time is the critical performance issue for **users** of a server. Utilization frequently is important to the **administrators** of a server. Maximum throughput is indicative of the performance *bottleneck*, and may not be a concern. While all of these characteristics are interrelated, the following summarizes server performance:

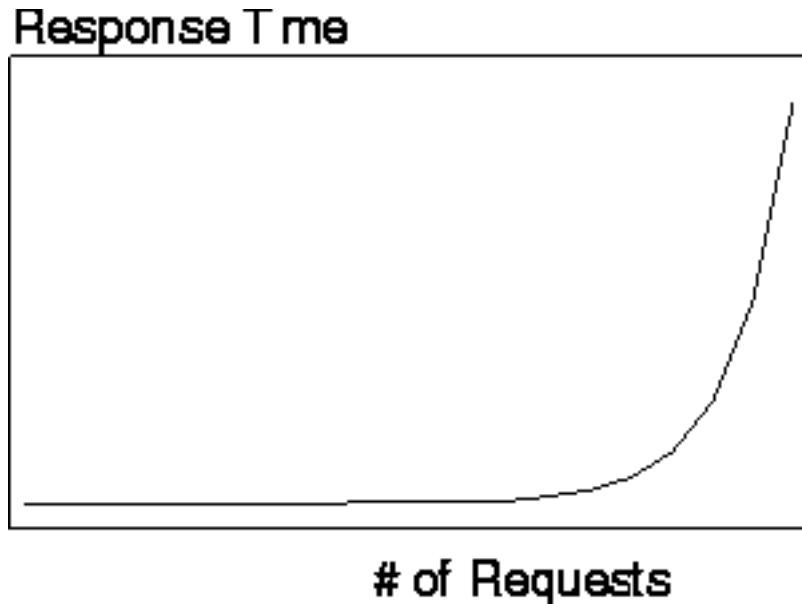
- Every computing server has a bottleneck that governs performance: **throughput**.
- When server utilization increases, response time degrades.

In many servers, capacity is considerable, and is not an issue with users. In others, it is the primary performance concern. Response time is critical. One of the most important questions for administrators is: *How much can the server be degraded (by adding users, increasing utilization) before users begin objecting?*

Introduction to client/server performance: The performance characteristics of a client/server environment are different than those of centralized environments. This is because client/server applications are split between the client and the server. The client and server communicate by sending and receiving requests and messages. This model is far different than that for a centralized environment. In that environment, a program calls the CPU, and the memory and disk drives are fully dedicated.

Instead, when a client requests processing time and data from the server, it transmits the request on the network. The request travels to the server and waits in a queue until the server is able to process it. The performance characteristics of this type of architecture degrade exponentially as the number of requests increase. In other words, response times increase gradually as more requests are made, but then increase dramatically at some point, which is known as the “knee of the curve.” This concept is illustrated by the

following graph:



It is important to determine this point at which performance begins to degrade significantly. The point can vary with every client/server installation.

The following is a suggested guideline for client/server operations: *Communicate with the server only when necessary, and in as few data transmissions as possible.* Opening a file and reading one record at a time often results in problems for client-server projects and tools.

The performance architecture of the iSeries Access for Windows ODBC driver: For the iSeries Access for Windows ODBC driver, all of the internal data flows between the client and the server are chained together, and transmit only when needed. This reduces server utilization because communications-layer resources are allocated only once. Response times improve correspondingly.

These types of enhancements are transparent to the user. However, there are some enhancements which are exposed on the iSeries Access for Windows ODBC Setup dialog. Look at the online help on the Performance tab of the setup GUI or refer to the Performance options on the Connection String keywords descriptions for more information. A few of these performance options are also discussed in more detail at the following links:

- “Selecting a stringent level of commitment control”
- “Fine-tuning record blocking” on page 605
- “Using Extended Dynamic SQL” on page 605

ODBC registry settings: When you edit the configuration parameters in the ODBC registry, the iSeries Access for Windows ODBC driver also is configured. This registry file is in the directory where Windows is installed on your server. Do not edit the registry directly. Instead, tune iSeries Access for Windows ODBC performance through the iSeries Access for Windows ODBC Setup dialog.

ODBC registry settings topics:

- “Selecting a stringent level of commitment control”
- “Fine-tuning record blocking” on page 605
- “Using Extended Dynamic SQL” on page 605

Selecting a stringent level of commitment control: Do not use commitment control unnecessarily. The overhead that is associated with locking not only increases utilization, but also reduces concurrency. However, if your application is not read-only, commitment control *may* be required. A common alternative is

to use **optimistic locking**. Optimistic locking involves issuing explicit UPDATES by using a WHERE clause that uniquely determines a particular record. Optimistic locking ensures that the record does not change after it is retrieved.

Many third-party tools use this approach, which is why they often require a unique index to be defined for updatable tables. This allows the record update to be made by fully qualifying the entire record contents. Consider the following example:

```
UPDATE table SET C1=new_val1, C2=new_val2, C3=new_val3
WHERE C1=old_val1 AND C2=old_val2 AND C3=old_val3
```

This statement would guarantee that the desired row is accurately updated, but only if the table contained only three columns, and each row was unique. A better-performing alternative would be:

```
UPDATE table SET C1=new_val1, C2=new_val2, C3=CURRENT_TIMESTAMP
WHERE C3=old_timestamp
```

This only works, however, if the table has a timestamp column that holds information on when the record was last updated. Set the new value for this column to CURRENT_TIMESTAMP to guarantee row uniqueness.

Note: This technique does not work with any object model that uses automation data types (for example, Visual Basic, Delphi, scripting languages). The variant DATE data type has a timestamp precision of approximately one millisecond. The iSeries server timestamp is either truncated or rounded off, and the WHERE clause fails.

If commitment control is required, use the lowest level of record locking possible. For example, use ***CHG:** over ***CS** when possible, and never use ***ALL** when ***CS** provides what you require.

For more information on commitment control:

See the DB2 Universal Database for iSeries and DB2 Universal Database for iSeries books online topics under the **Database and File Systems** heading in the iSeries Information Center.

Fine-tuning record blocking: **Record blocking** is a technique that significantly reduces the number of network flows. It does this by returning a *block* of rows from the server on the first FETCH request for a cursor. Subsequent FETCH requests are retrieved from the local block of rows, rather than going to the server each time. This technique dramatically increases performance when it is properly used. The default settings should be sufficient for most situations.

A change to one of the record-blocking parameters can make a significant difference when the performance of your environment is approaching the exponential threshold that is illustrated in “Introduction to client/server performance” on page 603. For example, assume that an environment has *n* decision-support clients doing some amount of work with large queries, typically returning 1 MB of data.

At the opposite extreme is a scenario where users consistently ask for large amounts of data, but typically never examine more than a few rows. The overhead of returning 32KB of rows when only a few are needed could degrade performance. Setting the *BLOCKSIZE* or *BlockSizeKB* connection string keyword to a lower value, setting the *BLOCKFETCH* connection string keyword to 0 (Use ODBC blocking) or disabling record blocking altogether, might actually increase performance.

It is important to note that, as always in client/server, performance results may vary. You might make changes to these parameters and not realize any difference. This may indicate that your performance bottleneck is not the client request queue at the server. This parameter gives you one more tool to use when your users start objecting.

Using Extended Dynamic SQL: Traditional SQL interfaces used an embedded SQL approach. SQL statements were placed directly in an application’s source code, along with high-level language statements written in C, COBOL, RPG, and other programming languages. The source code then was precompiled, which translated the SQL statements into code that the subsequent compile step could process. This

method sometimes was referred to as **static SQL**. One performance advantage to this approach is that SQL statements were optimized in advance, rather than at runtime while the user was waiting.

ODBC, however, is a **call level interface** (CLI) that uses a different approach. Using a CLI, SQL statements are passed to the database management system (DBMS) within a parameter of a runtime API. Because the text of the SQL statement is never known until runtime, the optimization step must be performed each time an SQL statement is run. This approach commonly is referred to as **dynamic SQL**.

The use of this feature (which is enabled by default) not only can improve response times, but can improve dramatically server utilization. This is because optimizing SQL queries can be costly, and performing this step only once always is advantageous. This works well with a unique feature of DB2 UDB for iSeries. Unlike other DBMSs, it ensures that statements which are stored in packages are kept up-to-date in terms of optimization, without administrator intervention. Even if a statement was prepared for the first time weeks or months ago, DB2 UDB for iSeries automatically regenerates the access plan when it determines that sufficient database changes require reoptimization.

Performance considerations of common end-user tools

Having an ODBC driver that is optimally tuned is only part of the performance equation. The other part is the tools that are used; whether they are used simply to query the data, or to build complex programs.

Some of the more common tools include:

- Crystal Services Crystal Reports Professional
- Cognos Impromptu
- Gupta SQL Windows
- IBM Visualizer for Windows
- Lotus Approach
- Lotus Notes®
- Notes Pump
- Microsoft Access
- Microsoft Internet Information Server
- Microsoft SQL Server
- Microsoft Visual Basic
- Powersoft PowerBuilder

There are many more tools available than are on this list, and every tool in the marketplace has its own strengths, weaknesses, and performance characteristics. But most have one thing in common: support for ODBC database servers. However, because ODBC serves as a common denominator for various database management systems, and because there are subtle differences from one ODBC driver to the next, many tool providers write to the more common ODBC and SQL interfaces. By doing this, they avoid taking advantage of a unique characteristic of a particular database server. This may ease programming efforts, but it often degrades overall performance.

Examples of ODBC performance-degrading tools:

“Examples: Common tool behaviors that degrade ODBC performance”

Examples: Common tool behaviors that degrade ODBC performance: The following examples demonstrate performance problems that are associated with writing SQL and ODBC calls that do NOT take advantage of a unique feature of a particular ODBC driver or the server database management system.

To view the examples:

- “Example: Query tool A”
- “Example: Query tool B” on page 607
- “Example: Query tool C” on page 608

Example: Query tool A: Query Tool A makes the following ODBC calls to process SELECT statements:

```

SQLExecDirect("SELECT * FROM table_name")

WHILE there_are_rows_to_fetch DO

    SQLFetch()
    FOR every_column DO
        SQLGetData( COLn )
    END FOR
    ...process the data

END WHILE

```

This tool does not make use of ODBC bound columns, which can help performance. A faster way to process this is as follows:

```

SQLExecDirect("SELECT * FROM table_name")
FOR every_column DO
    SQLBindColumn( COLn )
END FOR

WHILE there_are_rows_to_fetch DO
    SQLFetch()
    ...process the data
END WHILE

```

If a table contained one column, there would be little difference between the two approaches. But for a table with a 100 columns, you end up with 100 times as many ODBC calls in the first example, *for every row fetched*. You also can optimize the second scenario because the target data types specified by the tool will not change from one FETCH to the next, like they could change with each **SQLGetData** call.

Example: Query tool B: Query tool B allows you to update a spreadsheet of rows and then send the updates to the database. It makes the following ODBC calls:

```

FOR every_row_updated DO

    SQLAllocHandle(SQL_HANDLE_STMT)
    SQLExecDirect("UPDATE...SET COLn='literal'...WHERE COLn='oldval'...")
    SQLFreeHandle( SQL_HANDLE_STMT )

END LOOP

```

The first thing to note is that the tool performs a statement allocation-and-drop for every row. Only one allocate statement is needed, and the free-statement call could be changed to `SQLFreeStmt(SQL_CLOSE)` after each **SQLExecDirect**. This change would save the overhead of creating and destroying a statement handle for every operation. Another performance concern is the use of SQL with literals instead of with parameter markers. The **SQLExecDirect()** call causes an **SQLPrepare** and **SQLExecute** every time. A faster way to perform this operation would be as follows:

```

SQLAllocHandle(SQL_HANDLE_STMT)
SQLPrepare("UPDATE...SET COL1=?...WHERE COL1=?...")
SQLBindParameter( new_column_buffers )
SQLBindParameter( old_column_buffers )
FOR every_row_updated DO

    ...move each rows data into the SQLBindParameter buffers
    SQLExecute()
    SQLFreeHandle( SQL_HANDLE_STMT )

END LOOP

```

These sets of ODBC calls will outperform the original set by a large factor when you are using the iSeries Access for Windows ODBC driver. The server CPU utilization will decrease to 10 percent of what it was, which pushes the scaling threshold out a lot farther.

Example: Query tool C: **Worst-case scenario**

Query tool C allows complex decision support-type queries to be made by defining complex query criteria with a point-and-click interface. You might end up with SQL that looks like this for a query:

```
SELECT A.COL1, B.COL2, C.COL3 , etc...
FROM A, B, C, etc...
WHERE many complex inner and outer joins are specified
```

That you did not have to write this complex query is advantageous, but beware that your tool may not actually process this statement. For example, one tool might pass this statement directly to the ODBC driver, while another splits up the query into many individual queries, and processes the results at the client, like this:

```
SQLExecDirect("SELECT * FROM A")
SQLFetch() all rows from A
SQLExecDirect("SELECT * FROM B")
SQLFetch() all rows from B
```

Process the first join at the client

```
SQLExecDirect("SELECT * FROM C")
SQLFetch() all rows from C
```

Process the next join at the client

```
.
.
.
```

And so on...

This approach can lead to excessive amounts of data being passed to the client, which will adversely affect performance. In one real-world example, a programmer thought that a 10-way inner/outer join was being passed to ODBC, with four rows being returned. What actually was passed, however, was 10 simple SELECT statements and all the FETCHes associated with them. The net result of four rows was achieved only after *81,000* ODBC calls were made by the tool. The programmer initially thought that ODBC was responsible for the slow performance, until the ODBC trace was revealed.

SQL performance

Good application design includes the efficient use of machine resources. To run in a manner that is acceptable to the end user, an application program must be efficient in operation, and must run with adequate response time.

“SQL performance general considerations”

Shows you when to consider performance, what resources to optimize, and how to design for performance.

“Database design” on page 609

Describes general iSeries database design and how it affects SQL performance.

“Optimizer” on page 613

Optimizer is the facility that decides how to gather data that should be returned to the program. This topic covers some of the techniques and rules that are used by Optimizer.

SQL performance general considerations: Performance of SQL in application programs is important to ALL server users, because inefficient usage of SQL can waste server resources.

The primary goal in using SQL is to obtain the correct results for your database request, and in a timely manner.

Before you start designing for performance, review the following considerations:

When to consider performance:

- Database with over 10,000 rows - Performance impact: **noticeable**
- Database with over 100,000 rows - Performance impact: **concern**
- When repetitively using complex queries
- When using multiple work stations with high transaction rates

What resource to optimize:

- I/O usage
- CPU usage
- Effective usage of indexes
- OPEN/CLOSE performance
- Concurrency (COMMIT)

How to design for performance:

Database design:

- Table structure
- Indexes
- Table data management
- Journal management

Application design:

- Structure of programs involved

Program design:

- Coding practices
- Performance monitoring

The *SQL Reference* book contains additional information. You can view an HTML online version of the book, or print a PDF version, from the DB2 Universal Database for iSeries books online iSeries Information Center topic.

Database design: The following topics help you to:

- Determine what tables you require in your database
- Understand the relationship between those tables

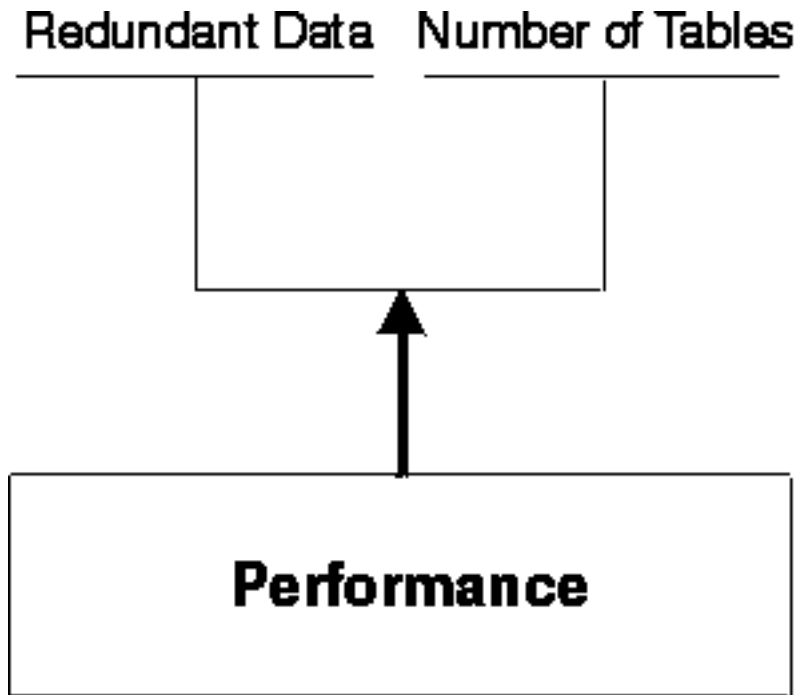
Database design topics:

- “Normalization”
- “Table size” on page 612
- “Using indexes” on page 612
- “Matching attributes of join fields” on page 613

Normalization: Several available design methods allow you to design technically correct databases, and effective relational database structure. Some of these methods are based on a design approach called normalization. Normalization refers to the reduction or elimination of storing redundant data. The primary objective of normalization is to avoid problems that are associated with updating redundant data.

However, this design approach of normalization (for example, 3NF–3rd Normal Form), may result in large numbers of tables. If there are numerous table join operations, SQL performance may be reduced. Consider overall SQL performance when you design databases. Balance the amount of redundant data with the number of tables that are not fully normalized.

The following graphic illustrates that the proportion of redundant data to the number of tables affects performance:



Minimize the use of code tables when little is gained from their use. For example, an employee table contains a JOBCODE column, with data values 054, 057, and so forth. This table must be joined with another table to translate the codes to Programmer, Engineer, and so on. The cost of this join could be quite high compared to the savings in storage and potential update errors resulting from redundant data.

For example:

Normalized data form:

EMPLOYEE Table

Employee No	Jobcode
00010	057
00020	054
00030	057
.	.

JOB CODE Table

Jobcode	Job Title
054	Programmer
057	Engineer
.	.
.	.

Redundant data form:

EMPLOYEE Table

Emp oyeē No	Job Tit e
00010	Eng neēr
00020	P·ogrammer
00030	Eng neēr
.	.

The set level (or mass operation) nature of SQL significantly lessens the danger of a certain redundant data form. For example, the ability to update a set of rows with a single SQL statement greatly reduces this risk. In the following example, the job title **Engineer** must be changed to **Technician** for all rows that match this condition.

Example: Use SQL to update JOBTITLE:

```
UPDATE EMPLOYEE
SET JOBTITLE = "Technician"
WHERE JOBTITLE = "Engineer"
```

Table size: The size of the tables that your application program accesses has a significant impact on the performance of the application program. Consider the following:

Large row length:

For sequentially accessed tables that have a large row length because of many columns (100 or more), you may improve performance by dividing the tables into several smaller ones, or by creating a view. This assumes that your application is not accessing all of the columns. The main reason for the better performance is that I/O may be reduced because you will get more rows per page. Splitting the table will affect applications that access all of the columns because they will incur the overhead of joining the table back together again. You must decide where to split the table based on the nature of the application and frequency of access to various columns.

Large number of rows:

If a table has a large number of rows, construct your SQL statements so that the "Optimizer" on page 613 uses an index to access the table. The use of indexes is very important for achieving the best possible performance.

Using indexes: The use of indexes can improve significantly the performance of your applications. This is because the "Optimizer" on page 613 uses them for performance optimization. Indexes are created in five different ways:

- CREATE INDEX (in SQL)
- CRTPF, with key
- CRTLF, with key

- CRTLF, as join logical file
- CRTLF, with select/omit specifications, without a key, and without dynamic selection (DYNSTL).

Indexes are used to enable row selection by means of index-versus-table scanning, which is usually slower. Table scanning sequentially processes all rows in a table. If a permanent index is available, building a temporary index can be avoided. Indexes are required for:

- Join tables
- ORDER BY
- GROUP BY

Indexes will be created, if no permanent index exists.

Manage the number of indexes to minimize the extra server cost of maintaining the indexes during update operations. Below are general rules for particular types of tables:

Primarily read-only tables:

Create indexes over columns as needed. Consider creating an index only if a table is greater than approximately 1,000 rows or is going to be used with ORDER BY, GROUP BY, or join processing. Index maintenance could be costlier than occasionally scanning the entire table.

Primarily read-only table, with low update rate:

Create indexes over columns as needed. Avoid building indexes over columns that are updated frequently. INSERT, UPDATE, and DELETE will cause maintenance to all indexes related to the table.

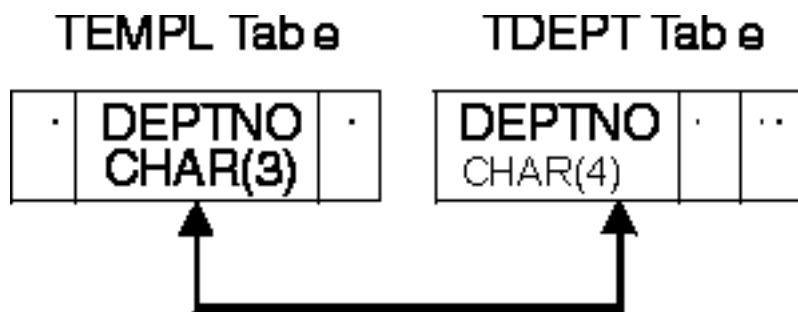
High update-rate tables:

Avoid creating many indexes. An example of a table that has a high update rate is a logging or a history table.

Matching attributes of join fields: Columns in tables that are joined should have identical attributes: the same column length, same data type (character, numeric), and so forth. Nonidentical attributes result in temporary indexes being built, even though indexes over corresponding columns may exist.

In the following example, **join** will build a temporary index and ignore an existing one:

```
SELECT EMPNO, LASTNAME, DEPTNAME
FROM TEMPL, TDEPT
WHERE TEMPL.DEPTNO = TDEPT.DEPTNO
```



Optimizer: Optimizer is an important module of the OS/400 Query component because it makes the key decisions for good database performance. Its main objective is to find the most efficient access path to the data.

Query optimization is a trade-off between the time spent to select a query implementation and the time spent to run it. Query optimization must handle the following distinct user needs:

- Quick interactive response

- Efficient use of total-machine resources

In deciding how to access data, Optimizer does the following:

- Determines possible implementations
- Picks the optimal implementation for the OS/400 Query component to execute

Optimizer topics:

- “Cost estimation”
- “Optimizer decision-making rules” on page 615

Cost estimation: At runtime, the Optimizer chooses an optimal access method for the query by calculating an implementation cost based on the current state of the database. The Optimizer models the access cost of each of the following:

- Reading rows directly from the table (dataspace scan processing)
- Reading rows through an access path (using either key selection or key positioning)
- Creating an access path directly from the dataspace
- Creating an access path from an existing access path (index-from-index)
- Using the query sort routine (if conditions are satisfied)

The cost of a particular method is the sum of:

- The start-up cost
- The cost associated with the given optimization mode. The OPTIMIZE FOR n ROWS clause indicates to the query Optimizer the optimization goal to be achieved. The Optimizer can optimize SQL queries with one of two goals:
 1. Minimize the time required to retrieve the first buffer of rows from the table. This goal biases the optimization towards not creating an index.

Note: This is the default if you do not use OPTIMIZE FOR n ROWS.

Either a data scan or an existing index is preferred. This mode can be specified by:

The OPTIMIZE FOR n ROWS allowing the users to specify the number of rows they expect to retrieve from the query.

The Optimizer using this value to determine the percentage of rows that will be returned and optimizes accordingly. A small value instructs the Optimizer to minimize the time required to retrieve the first n rows.

2. Minimize the time to process the whole query assuming that all selected rows are returned to the application. This does not bias the Optimizer to any particular access method. Specify this mode by using OPTIMIZE FOR n ROWS, which allows the users to specify the number of rows they expect to retrieve from the query.

The Optimizer uses this value to determine the percentage of rows that will be returned and optimizes accordingly. A value greater than or equal to the expected number of resulting rows instructs the Optimizer to minimize the time required to run the entire query.

- The cost of any access path creations.
- The cost of the expected number of page faults to read the rows and the cost of processing the expected number of rows.

Page faults and number of rows processed may be predicted by statistics the Optimizer obtains from the database objects, including:

- Table size
- Row size
- Index size
- Key size

A weighted measure of the expected number of rows to process. This is based on what the relational operators in the row selection predicates (default filter factors) are likely to retrieve:

- 10% for equal
- 33% for less-than, greater-than, less-than-equal-to, or greater-than-equal-to
- 90% for not equal
- 25% for BETWEEN range
- 10% for each IN list value

Key range estimate is a method that the Optimizer uses to gain more accurate estimates of the number of expected rows that are selected from one or more selection predicates. The Optimizer estimates by applying the selection predicates against the left-most keys of an existing index. The **default filter factors** then can be further refined by the estimate based on the key range. If the left-most keys in an index match columns that are used in row-selection predicates, use that index to estimate the number of keys that match the selection criteria. The estimate of the number of keys is based on the number of pages and key density of the machine index. It is performed without actually accessing the keys. Full indexes over columns that are used in selection predicates can significantly help optimization.

Optimizer decision-making rules: In performing its function, Optimizer uses a general set of guidelines to choose the best method for accessing data. Optimizer does the following:

- Determines the default filter factor for each predicate in the selection clause.
- Extracts attributes of the table from internally stored information.
- Performs an estimate key range to determine the true filter factor of the predicates when the selection predicates match the left-most keys of an index.
- Determines the cost of creating an index over a table if an index is required.
- Determines the cost of using a sort routine if selection conditions apply and an index is required.
- Determines the cost of dataspace scan processing if an index is not required.
- For each index available, in the order of most recently created to oldest, Optimizer does the following until its time limit is exceeded:
 - Extracts attributes of the index from internally stored statistics.
 - Determines if the index meets the selection criteria.
 - Determines the cost of using the index using the estimated page faults and the predicate filter factors to help determine the cost.
 - Compares the cost of using this index with the previous cost (current best).
 - Selects the cheapest one.
 - Continues to search for best index until time out or no more indexes.

The time limit factor controls how much time is spent choosing an implementation. It is based on how much time has been spent and the current best implementation cost found. Dynamic SQL queries are subject to Optimizer time restrictions. Static SQL queries optimization time is not limited.

For small tables, the query Optimizer spends little time in query optimization. For large tables, the query Optimizer considers more indexes. Generally, Optimizer considers five or six indexes (for each table of a join) before running out of optimization time.

ODBC blocked insert statement

The blocked **INSERT** statement provides a means to insert multiple rows with a single **SQLExecute** request. For performance, it provides the one of the best ways to populate a table, at times providing a tenfold performance improvement over the next best method.

The three forms of INSERT statements that can be executed from ODBC are:

- INSERT with VALUES using constants

- INSERT with VALUES using parameter markers
- blocked INSERT

The INSERT with VALUES using constants statement is the least efficient method of performing inserts. For each request, a single INSERT statement is sent to the server where it is prepared, the underlying table is opened, and the record is written.

Example:

```
INSERT INTO TEST.TABLE1 VALUES('ENGINEERING',10,'JONES','BOB')
```

The INSERT with VALUES using parameter markers statement performs better than the statement that uses constants. This form of the INSERT statement allows for the statement to be prepared only once and then reused on subsequent executions of the statement. It also allows the table on the server to remain open, thus removing the overhead of opening and closing the file for each insert.

Example:

```
INSERT INTO TEST.TABLE1 VALUES (?, ?, ?, ?)
```

The blocked INSERT statement most efficiently performs inserts into a table when multiple records can be cached on the client and sent at once. The advantages with blocked INSERT are:

- The data for multiple rows is sent in one communication request rather than one request per row.
- The server has an optimized path built into the database support for blocked INSERT statements.

Example:

```
INSERT INTO TEST.TABLE1 ? ROWS VALUES (?, ?, ?, ?)
```

The INSERT statement has additional syntax that identifies it as a blocked INSERT. The "? ROWS" clause indicates that an additional parameter will be specified for this INSERT statement. It also indicates that the parameter will contain a row count that determines how many rows will be sent for that execution of the statement. The number of rows must be specified by means of the **SQLSetStmtAttr** API.

Note: With the V5R1 driver, you do not need to specify the "? ROWS" clause to iSeries servers. V4R5 iSeries servers added this support via PTFs SF64146 and SF64149.

To view examples of blocked insert calls from C:

See "Block insert and block fetch C example" on page 579

Catalog functions

Catalog functions return information about a data source's catalog.

To process ODBC **SQLTables** requests, logical files are built over the server cross reference file QADBXREF in library QSYS. QADBXREF is a database file for database-maintained cross-reference information that is part of the dictionary function for the server.

The following are the actions for **SQLTables** when **TableType** is set to the following:

NULL Selects all LOGICAL and PHYSICAL files and SQL TABLES and VIEWS.

TABLE

Selects all PHYSICAL files, and SQL TABLES that are not server files (cross reference or data dictionary).

VIEW Selects all LOGICAL files and SQL VIEWS that are not server files (cross reference or data dictionary).

SYSTEM TABLE

Selects all PHYSICAL and LOGICAL files and SQL VIEWS that are either server files or data dictionary files.

TABLE, VIEW

Selects all LOGICAL and PHYSICAL files and all SQL TABLES and VIEWS that are not server files or data dictionary files.

Non-relational files (files with more than one format) are not selected. Also not selected are indexes, flat files and IDDU-defined files.

The result sets returned by the catalog functions are ordered by table type. In addition to the TABLE and VIEW types, the iSeries server has the data source-specific type identifiers of PHYSICAL and LOGICAL files. The PHYSICAL type is handled as a TABLE, and the LOGICAL type is handled as a VIEW.

To process ODBC **SQLColumns** requests, a logical file is built over the server cross-reference file QADBIFLD in the QSYS library. This logical file selects all relational database files except for indexes. QADBIFLD is a database file for database-maintained cross-reference information that is part of the dictionary function for the server. Specifically, this includes database file column and field information.

For additional information:

Appendix G of the *SQL Reference* book contains additional information. View an HTML online version of the book, or print a PDF version, from the DB2 Universal Database for iSeries books online iSeries Information Center topic.

Exit programs

When you specify an **exit program**, the servers pass the following two parameters to the exit program before running your request:

- A 1-byte return code value.
- A structure containing information about your request. This structure is different for each of the exit points.

These two parameters allow the exit program to determine whether your request is allowed. If the exit program sets the return code to X'F0', the server rejects the request. If the return code is set to anything else, the server allows the request.

The same program can be used for multiple exit points. The program can determine what function is being called by looking at the data in the second parameter structure.

Use the Work with Registration Information (WRKREGINF) command to add your exit programs to the database exit points.

The database server has four different exit points defined:

QIBM_QZDA_INIT

called at server initiation

QIBM_QZDA_NDB1

called for native database requests

QIBM_QZDA_SQL1

called for SQL requests

QIBM_QZDA_ROI1

called for retrieving object information requests and SQL catalog functions

Note: This exit point is called less often than in V5R1 and earlier Client Access ODBC drivers. If you have an exit program that uses this exit point, verify that it still works as intended.

Exit programs-related topics:

- “Examples: User exit programs”
- “Exit program parameter formats” on page 624

Examples: User exit programs: The following examples do not show all of the programming considerations or techniques. Review the examples before you begin application design and coding.

- | • “Example: ILE C/400® user exit program for exit point QIBM_QZDA_INIT”
- | • “Example: CL user exit program for exit point QIBM_QZDA_INIT” on page 619
- | • “Example: ILE C/400 Program for exit point QIBM_QZDA_SQL1” on page 619
- | • “Example: ILE C/400 program for exit point QIBM_QZDA_ROI1” on page 621

Example: ILE C/400® user exit program for exit point QIBM_QZDA_INIT:

```
/*-----  
*           OS/400 Servers - Sample Exit Program  
*  
*   Exit Point Name       : QIBM_QZDA_INIT  
*  
*   Description           : The following ILE C/400 program handles  
*                           ODBC security by rejecting requests from  
*                           certain users.  
*                           It can be used as a shell for developing  
*                           exit programs tailored for your  
*                           operating environment.  
*  
*   Input                 : A 1-byte return code value  
*                           X'F0' server rejects the request  
*                           anything else server allows the request  
*                           Structure containing information about the  
*                           request. The format used by this program  
*                           is ZDAI0100.  
*-----*/  
/*-----  
*   Includes  
*-----*/  
#include <string.h>           /* string functions          */  
/*-----  
*   User Types  
*-----*/  
typedef struct {              /* Exit Point QIBM_QZDA_INIT format ZDAI0100 */  
    char User_profile_name[10]; /* Name of user profile calling server*/  
    char Server_identififier[10]; /* database server value (*SQL)      */  
    char Exit_format_name[8]; /* User exit format name (ZDAI0100)  */  
    long Requested_function; /* function being preformed (0)      */  
} ZDAI0100_fmt_t;  
  
/*-----  
-----*/  
  
/*=====  
*   Start of mainline executable code  
*=====*/  
int main (int argc, char *argv[])  
{  
    ZDAI0100_fmt_t input; /* input format record          */  
  
    /* copy input parm into structure          */  
    memcpy(&input, (ZDAI0100_fmt_t *)argv[2], 32);  
  
    if /* if user name is GUEST                */  
        ( memcmp(input.User_profile_name, "GUEST ", 10)==0 )  
    {  
        /* set return code to reject the request.          */  
        memcpy( argv[1], "0", 1);  
    }  
}
```



```

    }
    else /* else user is someone else */
    {
        /* set return code to allow the request. */
        memcpy( argv[1], "1", 1);
    }
} /* End of mainline executable code */

```

Example: CL user exit program for exit point QIBM_QZDA_INIT:

```

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* OS/400 Servers - Sample Exit Program */
/* */
/* Exit Point Name : QIBM_QZDA_INIT */
/* */
/* Description : The following Control Language program */
/* handles ODBC security by rejecting */
/* requests from certain users. */
/* It can be used as a shell for developing */
/* exit programs tailored for your */
/* operating environment. */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
PGM PARM(&STATUS &REQUEST)

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Program call parameter declarations */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
DCL VAR(&STATUS) TYPE(*CHAR) LEN(1) /* Accept/Reject indicator */
DCL VAR(&REQUEST) TYPE(*CHAR) LEN(34) /* Parameter structure */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Parameter declares */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
DCL VAR(&USER) TYPE(*CHAR) LEN(10) /* User profile name calling server*/
DCL VAR(&SRVID) TYPE(*CHAR) LEN(10) /* database server value (*SQL) */
DCL VAR(&FORMAT) TYPE(*CHAR) LEN(8) /* Format name (ZDAI0100) */
DCL VAR(&FUNC) TYPE(*CHAR) LEN(4) /* function being preformed (0) */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Extract the various parameters from the structure */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
CHGVAR VAR(&USER) VALUE(%SST(&REQUEST 1 10))
CHGVAR VAR(&SRVID) VALUE(%SST(&REQUEST 11 10))
CHGVAR VAR(&FORMAT) VALUE(%SST(&REQUEST 21 8))
CHGVAR VAR(&FUNC) VALUE(%SST(&REQUEST 28 4))

/*-----*/
/*-----*/

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Begin main program */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* set return code to allow the request. */
CHGVAR VAR(&STATUS) VALUE('1')

/* if user name is GUEST set return code to reject the request. */
IF (&USER *EQ 'GUEST') THEN ( +
    CHGVAR VAR(&STATUS) VALUE('0') )

EXIT:
ENDPGM

```

Example: ILE C/400 Program for exit point QIBM_QZDA_SQL1:

```

/*-----*/
* OS/400 Servers - Sample Exit Program
*

```

```

*   Exit Point Name       : QIBM_QZDA_SQL1
*
*   Description           : The following ILE C/400 program will
*                           reject any UPDATE request for user GUEST.
*                           It can be used as a shell for developing
*                           exit programs tailored for your
*                           operating environment.
*
*   Input                 : A 1-byte return code value
*                           X'F0' server rejects the request
*                           anything else server allows the request
*                           Structure containing information about the
*                           request. The format used by this program
*                           is ZDAQ0100.
*-----*/
/*-----
*   Includes
*-----*/
#include <string.h>           /* string functions           */
#include <stdio.h>           /* standard IO functions      */
#include <ctype.h>           /* type conversion functions   */
/*=====
*   Start of mainline executable code
*=====*/
main(int argc, char *argv[])
{
    long i;
    _Packed struct zdaq0100 {
        char name[10];
        char servid[10];
        char fmtid[8];
        long funcid;
        char stmtname[18];
        char cursname[18];
        char prepopt[2];
        char opnattr[2];
        char pkgname[10];
        char pkglib[10];
        short drdaind;
        char commitf;
        char stmttxt[512];
    } *sptr, stx;

/*-----
/* initialize return variable to indicate ok status */
strncpy(argv[1], "1", 1);

/*-----
/* Address parameter structure for SQL exit program and move local
/* parameters into local variables.
/* (note : this is not necessary to evaluate the arguments passed in).
/*-----
sptr = (_Packed struct zdaq0100 *) argv[2];

strncpy(stx.name, sptr->name, 10);
strncpy(stx.servid, sptr->servid, 10);
strncpy(stx.fmtid, sptr->fmtid, 8);
stx.funcid = sptr->funcid;
strncpy(stx.stmtname, sptr->stmtname, 18);
strncpy(stx.cursname, sptr->cursname, 18);
strncpy(stx.opnattr, sptr->opnattr, 2);
strncpy(stx.prepopt, sptr->prepopt, 2);
strncpy(stx.pkglib, sptr->pkglib, 10);
strncpy(stx.pkgname, sptr->pkgname, 10);
stx.drdaind = sptr->drdaind;
stx.commitf = sptr->commitf;

```

```

strncpy(stx.stmttxt, sptr->stmttxt, 512);

/*****
/* check for user GUEST and an UPDATE statement */
/* if found return an error */
/*****
if (! (strcmp(stx.name, "GUEST      ", 10)) )
{
    for (i=0; i<6; i++)
        stx.stmttxt[i] = toupper(stx.stmttxt[i]);

    if (! strcmp(stx.stmttxt, "UPDATE", 6) )
        /* Force error out of SQL user exit pgm */
        strncpy(argv[1], "0", 1);
    else;
}
return;
} /* End of mainline executable code */

/*-----*/
/*-----*/

/* initialize return variable to indicate ok status */
strncpy(argv[1], "1", 1);

/*****
/* Address parameter structure for SQL exit program and move local */
/* parameters into local variables. */
/* (note : this is not necessary to evaluate the arguments passed in). */
/*****
sptr = (_Packed struct zdaq0100 *) argv[2];

strncpy(stx.name, sptr->name, 10);
strncpy(stx.servid, sptr->servid, 10);
strncpy(stx.fmtid, sptr->fmtid, 8);
stx.funcid = sptr->funcid;
strncpy(stx.stmtname, sptr->stmtname, 18);
strncpy(stx.cursname, sptr->cursname, 18);
strncpy(stx.opnattr, sptr->opnattr, 2);
strncpy(stx.prepopt, sptr->prepopt, 2);
strncpy(stx.pkglib, sptr->pkglib, 10);
strncpy(stx.pkgname, sptr->pkgname, 10);
stx.drdaind = sptr->drdaind;
stx.commitf = sptr->commitf;
strncpy(stx.stmttxt, sptr->stmttxt, 512);

/*****
/* check for user GUEST and an UPDATE statement */
/* if found return an error */
/*****
if (! (strcmp(stx.name, "GUEST      ", 10)) )
{
    for (i=0; i<6; i++)
        stx.stmttxt[i] = toupper(stx.stmttxt[i]);

    if (! strcmp(stx.stmttxt, "UPDATE", 6) )
        /* Force error out of SQL user exit pgm */
        strncpy(argv[1], "0", 1);
    else;
}
return;
} /* End of mainline executable code */

```

Example: ILE C/400 program for exit point QIBM_QZDA_ROI1:

```

/*-----
*           OS/400 Servers - Sample Exit Program
*
*   Exit Point Name       : QIBM_QZDA_ROI1
*
*   Description           : The following ILE C/400 program logs all
*                           requests for catalog functions to the
*                           ZDALOG file in QGPL.
*                           It can be used as a shell for developing
*                           exit programs tailored for your
*                           operating environment.
*
*   Input                 : A 1-byte return code value
*                           X'F0' server rejects the request
*                           anything else server allows the request
*                           Structure containing information about the
*                           request. The format used by this program
*                           is ZDAR0100.
*
*   Dependencies         : The log file must be created using the
*                           following command:
*                           CRTPF FILE(QGPL/ZDALOG) RCDLEN(132)
*-----*/
/*-----
*   Includes
*-----*/
#include <recio.h>           /* record IO functions          */
#include <string.h>          /* string functions           */
/*-----
*   User Types
*-----*/
typedef struct {             /* Exit Point QIBM_QZDA_ROI1 format ZDAR0100 */
    char User_profile_name[10]; /* Name of user profile calling server*/
    char Server_identifier[10]; /* database server value (*RTV0BJINF) */
    char Exit_format_name[8];  /* User exit format name (ZDAR0100) */
    long Requested_function;   /* function being Requested          */
    char Library_name[20];     /* Name of library                  */
    char Database_name[36];    /* Name of relational database      */
    char Package_name[20];     /* Name of package                  */
    char File_name[256];       /* Name of file                      */
    char Member_name[20];      /* Name of member                   */
    char Format_name[20];       /* Name of format                   */
} ZDAR0100_fmt_t;

/*-----
*-----*/

/*=====
*   Start of mainline executable code
*=====*/
int main (int argc, char *argv[])
{
    _RFILE *file_ptr;         /* pointer to log file          */
    char output_record[132];  /* output log file record       */
    ZDAR0100_fmt_t input;     /* input format record          */
    /* set return code to allow the request.          */
    memcpy( argv[1], "1", 1);

    /* open the log file for writing to the end of the file          */
    if ( ( file_ptr = _Ropen("QGPL/ZDALOG", "ar") ) == NULL)
    {
        /* open failed          */
        return;
    }

    /* copy input parm into structure          */
    memcpy(&input, (ZDAR0100_fmt_t *)argv[2], 404);
}

```

```

switch /* Create the output record based on requested function */
      (input.Requested_function)
{
  case 0X1800: /* Retrieve library information */
    sprintf(output_record,
            "%10.10s retrieved library %20.20s",
            input.User_profile_name, input.Library_name);
    break;
  case 0X1801: /* Retrieve relational database information */
    sprintf(output_record,
            "%10.10s retrieved database %36.36s",
            input.User_profile_name, input.Database_name);
    break;
  case 0X1802: /* Retrieve SQL package information */
    sprintf(output_record,
            "%10.10s retrieved library %20.20s package %20.20s",
            input.User_profile_name, input.Library_name,
            input.Package_name);
    break;
  case 0X1803: /* Retrieve SQL package statement information */
    sprintf(output_record,
            "%10.10s retrieved library %20.20s package %20.20s statement info",
            input.User_profile_name, input.Library_name,
            input.Package_name);
    break;
/*-----*/
  case 0X1804: /* Retrieve file information */
    sprintf(output_record,
            "%10.10s retrieved library %20.20s file %40.40s",
            input.User_profile_name, input.Library_name, input.File_name);
    break;
  case 0X1805: /* Retrieve file member information */
    sprintf(output_record,
            "%10.10s retrieved library %20.20s member %20.20s file %40.40s",
            input.User_profile_name, input.Library_name,
            input.Member_name, input.File_name);
    break;
  case 0X1806: /* Retrieve record format information */
    sprintf(output_record,
            "%10.10s retrieved library %20.20s format %20.20s file %40.40s",
            input.User_profile_name, input.Library_name,
            input.Format_name, input.File_name);
    break;
  case 0X1807: /* Retrieve field information */
    sprintf(output_record,
            "%10.10s retrieved field info library %20.20s file %40.40s",
            input.User_profile_name, input.Library_name, input.File_name);
    break;
  case 0X1808: /* Retrieve index information */
    sprintf(output_record,
            "%10.10s retrieved index info library %20.20s file %40.40s",
            input.User_profile_name, input.Library_name, input.File_name);
    break;
  case 0X180B: /* Retrieve special column information */
    sprintf(output_record,
            "%10.10s retrieved column info library %20.20s file %40.40s",
            input.User_profile_name, input.Library_name, input.File_name);
    break;
  default : /* Unknown requested function */
    sprintf(output_record, "Unknown requested function");
    break;
} /* end switch statement */

/* write the output record to the file */

```

```

    _Rwrite(file_ptr, &output_record, 132);

    /* close the log file */
    _Rclose ( file_ptr );
} /* End of mainline executable code */

```

Exit program parameter formats: The exit points for native database and retrieving object information have two formats that are defined: QIBM_QZDA_SQL1 and QIBM_QZDA_SQL2. Depending on the type of function that is requested, one of the formats is used.

The QIBM_QZDA_SQL2 exit point is defined to run an exit point for certain SQL requests that are received for the database server. This exit point takes precedence over the QIBM_QZDA_SQL1 exit point. If a program is registered for the QIBM_QZDA_SQL2 exit point, it will be called, and a program for the QIBM_QZDA_SQL1 exit point will not be called.

Functions that cause the exit program to be called

- Prepare
- Open
- Execute
- Connect
- Create package
- Clear package
- Delete package
- Stream fetch
- Execute immediate
- Prepare and describe
- Prepare and execute or prepare and open
- Open and fetch
- Execute or open

Parameter fields and their descriptions for exit programs with different exit points and formats:

- “Parameter fields for exit point QIBM_QZDA_SQL2 format ZDAQ0200”
- “Parameter fields for exit point QIBM_QZDA_INIT format ZDAI0100” on page 626
- “Parameter fields for exit point QIBM_QZDA_NDB1 format ZDAD0100” on page 626
- “Parameter fields for exit point QIBM_QZDA_NDB1 format ZDAD0200” on page 627
- “Parameter fields for exit point QIBM_QZDA_SQL1 format ZDAQ0100” on page 628
- “Parameter fields for exit point QIBM_QZDA_ROI1 format ZDAR0100” on page 630
- “Parameter fields for exit point QIBM_QZDA_ROI1 format ZDAR0200” on page 631

Parameter fields for exit point QIBM_QZDA_SQL2 format ZDAQ0200: The following table shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_SQL2 with the ZDAQ0200 format:

Table 7. Exit point QIBM_QZDA_SQL2 format ZDAQ0200

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	The value is *SQLSRV for this exit point.
20	14	CHAR(8)	Format name	The user exit format name being used. For QIBM_QZDA_SQL1, the format name is ZDAQ0100.

Table 7. Exit point QIBM_QZDA_SQL2 format ZDAQ0200 (continued)

Offset		Type	Field	Description
Dec	Hex			
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: X'1800' - Prepare X'1803' - Prepare and describe X'1804' - Open/describe X'1805' - Execute X'1806' - Execute immediate X'1809' - Connect X'180C' - Stream fetch X'180D' - Prepare and execute X'180E' - Open and fetch X'180F' - Create package X'1810' - Clear package X'1811' - Delete package X'1812' - Execute or open
32	20	CHAR(18)	Statement name	Name of the statement used for the prepare or execute functions.
50	32	CHAR(18)	Cursor name	Name of the cursor used for the open function.
68	44	CHAR(2)	Prepare option	Option used for the prepare function.
70	46	CHAR(2)	Open attributes	Option used for the open function.
72	48	CHAR(10)	Extended dynamic package name	Name of the extended dynamic package.
82	52	CHAR(10)	Package library name	Name of the library for extended dyanmic SQL package.
92	5C	BINARY(2)	DRDA [®] indicator	0 - Connected to local RDB 1 - Connected to remote RDB
94	5E	CHAR(1)	Commitment control level	'A' - Commit *ALL 'C' - Commit *CHANGE 'N' - Commit *NONE 'S' - Commit *CS (cursor stability)
95	5F	CHAR(10)	Default SQL collection	Name of the default SQL collection used by the iSeries Database Server.
105	69	CHAR(129)	Reserved	Reserved for future parameters.
234	EA	BINARY(4)	SQL statement text length	Length of SQL statement text in the field that follows. The length can be a maximum of 32K.
238	EE	CHAR(*)	SQL statement text	Entire SQL statement.

Note: This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLSRC in library QSYSINC.

The QIBM_QZDA_INIT exit point is defined to run an exit program at server initiation. If a program is defined for this exit point, it is called each time the database server is initiated.

Parameter fields for exit point QIBM_QZDA_INIT format ZDAI0100: The following table shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_INIT using the ZDAI0100 format:

Table 8. Exit point QIBM_QZDA_INIT format ZDAI0100

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	The value is *SQL for this exit point.
20	14	CHAR(8)	Format name	The user exit format name being used. For QIBM_QZDA_INIT the format name is ZDAI0100.
28	1C	BINARY(4)	Requested function	The function being performed. The only valid value for this exit point is 0.
Note: This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLSRC in library QSYSINC.				

The QIBM_QZDA_NDB1 exit point is defined to run an exit program for native database requests for the database server. Two formats are defined for this exit point.

Functions that use format ZDAD0100:

- Create source physical file
- Create database file, based on existing file
- Add, clear, delete database file member
- Override database file
- Delete database file override
- Delete file

Note: Format ZDAD0200 is used when a request is received to add libraries to the library list.

Parameter fields for exit point QIBM_QZDA_NDB1 format ZDAD0100: The following table shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_NDB1 using the ZDAD0100 format:

Table 9. Exit point QIBM_QZDA_NDB1 format ZDAD0100

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For this exit point the value is *NDB.
20	14	CHAR(8)	Format name	The user exit format name being used. For the following functions, the format name is ZDAD0100.

Table 9. Exit point QIBM_QZDA_NDB1 format ZDAD0100 (continued)

Offset		Type	Field	Description
Dec	Hex			
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: X'1800' - Create source physical file X'1801' - Create database file, based on existing file X'1802' - Add database file member X'1803' - Clear database file member X'1804' - Delete database file member X'1805' - Override database file X'1806' - Delete database file override X'1807' - Create save file X'1808' - Clear save file X'1809' - Delete file
32	20	CHAR(128)	File name	Name of the file used for the requested function.
160	A0	CHAR(10)	Library name	Name of the library that contains the file.
170	AA	CHAR(10)	Member name	Name of the member to be added, cleared, or deleted.
180	B4	CHAR(10)	Authority	Authority to the created file
190	BE	CHAR(128)	Based on file name	Name of the file to use when creating a file based on an existing file.
318	13E	CHAR(10)	Based on library name	Name of the library containing the based on file
328	148	CHAR(10)	Override file name	Name of the file to be overridden
338	152	CHAR(10)	Override library name	Name of the library that contains the file to be overridden
348	15C	CHAR(10)	Override member name	Name of the member to be overridden
Note: This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLSRC in library QSYSINC.				

Parameter fields for exit point QIBM_QZDA_NDB1 format ZDAD0200: The following table shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_NDB1 by using the ZDAD0200 format:

Table 10. Exit point QIBM_QZDA_NDB1 format ZDAD0200

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For this exit point the value is *NDB.
20	14	CHAR(8)	Format name	The user exit format name being used. For the add to library list function the format name is ZDAD0200.

Table 10. Exit point QIBM_QZDA_NDB1 format ZDAD0200 (continued)

Offset		Type	Field	Description
Dec	Hex			
28	1C	BINARY(4)	Requested function	The function being performed. X'180C' - Add library list
32	20	BINARY(4)	Number of libraries	The number of libraries (the next field)
36	24	CHAR(10)	Library name	The library names for each library
Note: This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLESRC in library QSYSINC.				

The QIBM_QZDA_SQL1 exit point is defined to run an exit point for certain SQL requests that are received for the database server. Only one format is defined for this exit point.

Functions that use format ZDAD0200:

- Prepare
- Open
- Execute
- Connect
- Create package
- Clear package
- Delete package
- Execute immediate
- Prepare and describe
- Prepare and execute or prepare and open
- Open and fetch
- Execute or open

Parameter fields for exit point QIBM_QZDA_SQL1 format ZDAQ0100: The following table shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_SQL1 using the ZDAQ0100 format.

Table 11. Exit point QIBM_QZDA_SQL1 format ZDAQ0100

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For this exit point the value is *SQLSRV.
20	14	CHAR(8)	Format name	The user exit format name being used. For QIBM_QZDA_SQL1 the format name is ZDAQ0100.

Table 11. Exit point QIBM_QZDA_SQL1 format ZDAQ0100 (continued)

Offset		Type	Field	Description
Dec	Hex			
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: X'1800' - Prepare X'1803' - Prepare and describe X'1804' - Open/Describe X'1805' - Execute X'1806' - Execute immediate X'1809' - Connect X'180D' - Prepare and execute or prepare and open X'180E' - Open and fetch X'180F' - Create package X'1810' - Clear package X'1811' - Delete package X'1812' - Execute or open
32	20	CHAR(18)	Statement name	Name of the statement used for the prepare or execute functions.
50	32	CHAR(18)	Cursor name	Name of the cursor used for the open function.
68	44	CHAR(2)	Prepare option	Option used for the prepare function.
70	46	CHAR(2)	Open attributes	Option used for the open function.
72	48	CHAR(10)	Extended dynamic package name	Name of the extended dynamic SQL package.
82	52	CHAR(10)	Package library name	Name of the library for extended dynamic SQL package.
92	5C	BINARY(2)	DRDA indicator	0 - Connected to local RDB 1 - Connected to remote RDB
94	5E	CHAR(1)	Commitment control level	'A' - Commit *ALL 'C' - Commit *CHANGE 'N' - Commit *NONE 'S' - Commit *CS (cursor stability)
95	5F	CHAR(512)	First 512 bytes of the SQL statement text	First 512 bytes of the SQL statement
<p>Note: This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLSRC in library QSYSINC.</p>				

The QIBM_QZDA_ROI1 exit point is defined to run an exit program for the requests that retrieve information about certain objects for the database server. It is also used for SQL catalog functions.

This exit point has two formats defined.

Objects for which format ZDAR0100 is used to retrieve information:

- Field (or column)
- File (or table)

- File member
- Index
- Library (or collection)
- Record format
- Relational database (or RDB)
- Special columns
- SQL package
- SQL package statement

Objects for which format ZDAR0200 is used to retrieve information:

- Foreign keys
- Primary keys

Parameter fields for exit point QIBM_QZDA_ROI1 format ZDAR0100: The following table shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_ROI1 using the ZDAR0100 format.

Table 12. Exit point QIBM_QZDA_ROI1 format ZDAR0100

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For the database server the value is *RTVOBJINF.
20	14	CHAR(8)	Format name	The user exit format name being used. For the following functions, the format name is ZDAR0100.
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: X'1800' - Retrieve library information X'1801' - Retrieve relational database information X'1802' - Retrieve SQL package information X'1803' - Retrieve SQL package statement information X'1804' - Retrieve file information X'1805' - Retrieve file member information X'1806' - Retrieve record format information X'1807' - Retrieve field information X'1808' - Retrieve index information X'180B' - Retrieve special column information
32	20	CHAR(20)	Library name	The library or search pattern used when retrieving information about libraries, packages, package statements, files, members, record formats, fields, indexes, and special columns.

Table 12. Exit point QIBM_QZDA_ROI1 format ZDAR0100 (continued)

Offset		Type	Field	Description
Dec	Hex			
52	34	CHAR(36)	Relational database name	The relational database name or search pattern used to retrieve RDB information.
88	58	CHAR(20)	Package name	The package name or search pattern used to retrieve package or package statement information.
108	6C	CHAR(256)	File name (SQL alias name)	The file name or search pattern used to retrieve file, member, record format, field, index, or special column information.
364	16C	CHAR(20)	Member name	The member name or search pattern used to retrieve file member information.
384	180	CHAR(20)	Format name	The format name or search pattern used to retrieve record format information.
Note: This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLSRC in library QSYSINC.				

Parameter fields for exit point QIBM_QZDA_ROI1 format ZDAR0200: The following table shows parameter fields and their descriptions for the exit program called at exit point QIBM_QZDA_ROI1 using the ZDAR0200 format.

Table 13. Exit point QIBM_QZDA_ROI1 format ZDAR0200

Offset		Type	Field	Description
Dec	Hex			
0	0	CHAR(10)	User profile name	The name of the user profile that is calling the server.
10	A	CHAR(10)	Server identifier	For the database server the value is *RTVOBJINF.
20	14	CHAR(8)	Format name	The user exit format name being used. For the following functions, the format name is ZDAR0200.
28	1C	BINARY(4)	Requested function	The function being performed. This field contains one of the following: X'1809' - Retrieve foreign key information X'180A' - Retrieve primary key information
32	20	CHAR(10)	Primary key table library name	The name of the library that contains the primary key table used when retrieving primary and foreign key information.
42	2A	CHAR(128)	Primary key table name (alias name)	The name of the table that contains the primary key used when retrieving primary or foreign key information.
170	AA	CHAR(10)	Foreign key table library name	The name of the library that contains the foreign key table used when retrieving foreign key information.
180	64	CHAR(128)	Foreign key table name (alias name)	The name of the table that contains the foreign key used when retrieving foreign key information.

Table 13. Exit point QIBM_QZDA_ROI1 format ZDAR0200 (continued)

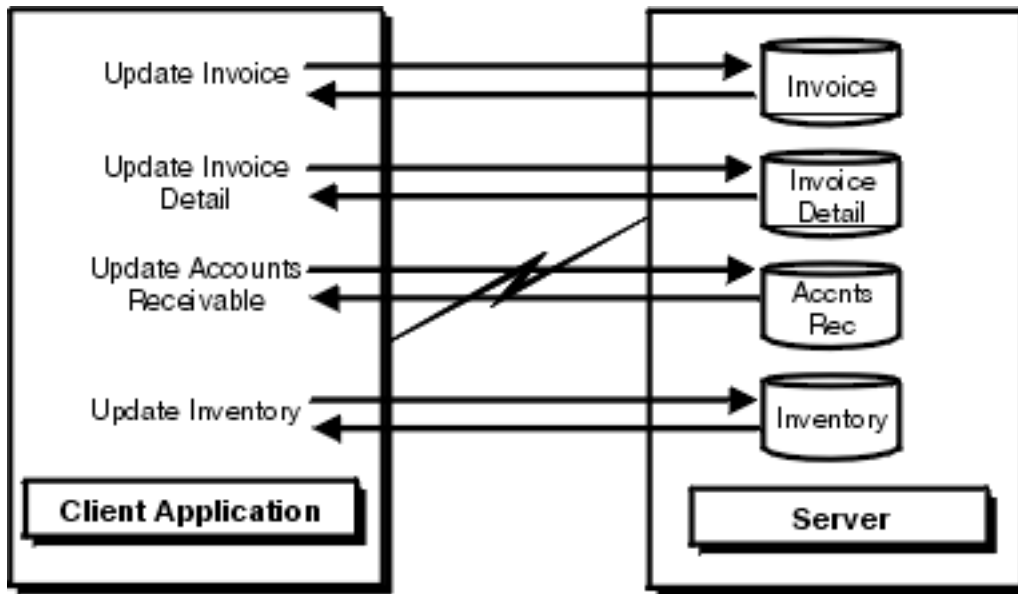
Offset		Type	Field	Description
Dec	Hex			
Note: This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLSRC in library QSYSINC.				

Stored procedures

Stored procedures commonly are used in client/server applications, especially in the area of online transaction processing (OLTP), since they can provide performance, transaction-integrity and security benefits.

For information regarding specific SQL commands that are used in the examples of stored procedures, see the *SQL Reference* book. View an HTML online version of the book, or print a PDF version, from the DB2 Universal Database for iSeries books online iSeries Information Center topic.

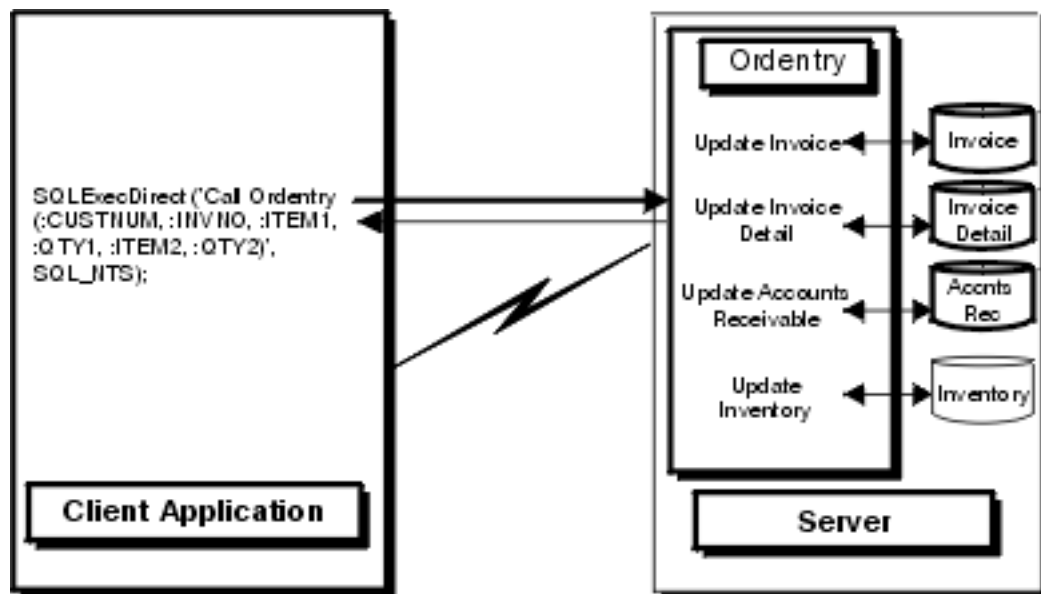
The illustration below shows an application where one transaction consists of four separate I/O operations, each that requires an SQL statement to be processed. In the client/server environment, this requires a minimum of eight messages between the server and the client, as shown. This can represent significant overhead, especially where the communication speed is slow (for example over a dial-up line), or where the turnaround speed for the connection is slow (for example over a satellite link).



Client/Server Application Without Stored Procedures

FIGURE 17.1

The following illustration shows the same transaction by a stored procedure on the server. As illustrated, the communications traffic has been reduced to a single message pair. There are additional benefits. For example, the procedure can arrange to send back only the data that is absolutely required (for example, just a few characters from a long column). A DB2 for OS/400[®] stored procedure can be any iSeries program, and does not have to use SQL for data access.



Client/Server Application With Stored Procedure

View examples of stored procedures:

- “Examples: Stored procedures”
- “Example: Visual C++ - Accessing and returning data by calling a stored procedure” on page 641
- “Example: Visual Basic - Accessing and returning data by calling a stored procedure” on page 642
- “Examples: RPG - Host code for ODBC stored procedures” on page 644
- “Tips: Running and calling iSeries stored procedures” on page 637

Examples: Stored procedures: View examples of stored procedures using the following:

- “Example: Running CL commands using SQL stored procedures and ODBC”
- “Example: Stored procedure calls from Visual Basic with return values” on page 634
- “Examples: Calling an iSeries stored procedure by using Visual Basic” on page 636

Example: Running CL commands using SQL stored procedures and ODBC: Stored procedure support provides a means to run iSeries server Control Language (CL) commands by using the SQL CALL statement.

Use CL commands when:

- Performing an override for files
- Initiating debug
- Using other commands that can affect the performance of subsequent SQL statements

The following examples show cases where a CL command is run on the iSeries server by using the CALL statement, which calls the program that processes CL commands. That program (QCMDEXC in library QSYS) expects two parameters:

1. A string that contains the command text to execute
2. A decimal (15,5) field that contains the length of the command text

The parameters must include these attributes for the command to be interpreted properly. The second parameter on the CALL statement must have characters explicitly specified for all places of the decimal (15,5) field.

In the following example, a C program on the PC is going to run an OVRDBF command that is 65 characters long (including embedded blanks). The text of the OVRDBF command is as follows:

```
OVRDBF FILE(TESTER) TOFILE(JMLIB/TESTER) MBR(NO2) OVRSCOPE(*JOB)
```

The code for performing this command by using ODBC APIs is as follows:

```
HSTMT hstmt;
SQLCHAR stmt[301];

rc = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
strcpy(stmt, "CALL QSYS.QCMDEXC('OVRDBF FILE(TESTER) TOFILE(MYLIB/");
strcat(stmt, "TESTER) MBR(NO2) OVRSCOPE(*JOB)',0000000064.00000)");
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
```

Statements now run against file MYLIB/TESTER will reference member number 2 rather than the first member.

Another CL command that is useful to run against a database server job is the STRDBG command. You do not have to call a stored procedure to run this command, though. There is an option on the Diagnostic tab of the DSN setup GUI on the Diagnostic tab that will automatically run the STRDBG command during the connection attempt.

Example: Stored procedure calls from Visual Basic with return values: Visual Basic is able to call external functions that are found in a DLL. Since all ODBC drivers are DLLs, Visual Basic can be used to code directly to the ODBC APIs. By coding directly to the ODBC APIs a Visual Basic application can call an iSeries server stored procedure and return result values. See “Coding directly to ODBC APIs” on page 568 for more information.

The following example of Visual Basic source code shows how to call an iSeries server stored procedure and then retrieve the returned values into Visual Basic variables.

```
'*****
' *
' * Because of the way Visual Basic stores and manages the String data *
' * type, it is recommended that you use an array of Byte data type *
' * instead of a String variable on the SQLBindParameter API. *
' *
' *
'*****

Dim sTemp As String
Custnum As Integer
Dim abCustname(34) As Byte
Dim abAddress(34) As Byte
Dim abCity(24) As Byte
Dim abState(1) As Byte
Dim abPhone(14) As Byte
Dim abStatus As Byte
Dim RC As Integer
Dim nullx As Long 'Used to pass null pointer, not pointer to null
Dim lpSQL_NTS As Long 'Used to pass far pointer to SQL_NTS
Static link(7) As Long 'Used as an array of long pointers to the size
'each parameter which will be bound

'*****
' *
' * Initialize the variables needed on the API calls *
' *
' *
'*****

link(1) = 6
link(2) = Ubound(abCustname) +1
link(3) = Ubound(abAddress) +1
```



```

link(4) = Ubound(abCity) +1
link(5) = Ubound(abState) +1
link(6) = Ubound(abPhone) +1
link(7) = 1

RC = 0
nullx = 0
lpSQL_NTS = SQL_NTS      ' -3 means passed as sz string

'*****
'*
'* Create the procedure on the iSeries. This will define the
'* procedure's name, parameters, and how each parameter is passed.
'* Note: This information is stored in the server catalog tables and
'* and only needs to be executed one time for the life of the stored
'* procedure. It normally would not be run in the client application.
'*
'*
'*****

sTemp = "Create Procedure Storedp2 (:Custnum in integer, "
sTemp = sTemp & ":Custname out char(35), :Address out char(35),"
sTemp = sTemp & ":City out char(25), :State out char(2),"
sTemp = sTemp & ":Phone out char(15), :Status out char(1))"
sTemp = sTemp & "(External name rastest.storedp2 language cobol General)"

RC = SQLExecDirect(Connection.hstmt, sTemp, Len(sTemp))

'Ignore error assuming that any error would be from procedure already
'created.

'*****
'*
'* Prepare the call of the procedure to the iSeries.
'* For best performance, prepare the statement only one time and
'* execute many times.
'*
'*
'*****

sTemp = "Call storedp2(?, ?, ?, ?, ?, ?, ?)"
RC = SQLPrepare(Connection.hstmt, sTemp, Len(sTemp))

If (RC <> SQL_SUCCESS) Then
    DescribeError Connection.hdbc, Connection.hstmt
    frmMain.Status.Caption = "Error on SQL_Prepere " & RTrim$(Tag)
End If

'*****
'*
'* Bind all of the columns passed to the stored procedure. This will
'* set up the variable's data type, input/output characteristics,
'* length, and initial value.
'* The SQLDescribeParam API can optionally be used to retrieve the
'* parameter types.
'*
'*
'* To properly pass an array of byte to a stored procedure and receive
'* an output value back, you must pass the first byte ByRef.
'*
'*
'*****

RC = SQLBindParameter(Connection.hstmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT, _
    SQL_NUMERIC, 6, 0, Custnum, 6, link(1))

RC = SQLBindParameter(Connection.hstmt, 2, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
    SQL_CHAR, 35, 0, abCustname(0), Ubound(abCustname)+1, link(2))
RC = SQLBindParameter(Connection.hstmt, 3, SQL_PARAM_OUTPUT, SQL_C_CHAR, _

```

```

SQL_CHAR, 35, 0, abAddress(0), UBound(abAddress)+1, link(3))
RC = SQLBindParameter(Connection.hstmt, 4, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
SQL_CHAR, 25, 0, abCity(0), UBound(abCity)+1, link(4))
RC = SQLBindParameter(Connection.hstmt, 5, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
SQL_CHAR, 2, 0, abState(0), UBound(abState)+1, link(5))
RC = SQLBindParameter(Connection.hstmt, 6, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
SQL_CHAR, 15, 0, abPhone(0), UBound(abPhone)+1, link(6))
RC = SQLBindParameter(Connection.hstmt, 7, SQL_PARAM_OUTPUT, SQL_C_CHAR, _
SQL_CHAR, 1, 0, abStatus, 1, link(7))

```

```

'*****
' *
' * The Prepare and Bind only needs to be execute once. The Stored
' * procedure can now be called multiple times by just changing the data
' *
'*****
Do While

```

```

'*****
' * Read in a customer number
' *
'*****

```

```

Custnum = Val(input.text)

```

```

'*****
' *
' * Execute the call of the procedure to the iSeries.
' *
'*****

```

```

RC = SQLExecute(Connection.hstmt)
frmMain.Status.Caption = "Ran Stored Proc" & RTrim$(Tag)

```

```

If (RC <> SQL_SUCCESS) Then
    DescribeError Connection.hdbc, Connection.hstmt
    frmMain.Status.Caption = "Error on Stored Proc Execute " & RTrim$(Tag)
End If

```

```

'*****
' *
' * Set text labels to display the output data
' * You must convert the array of Byte back to a String
' *
'*****

```

```

lblCustname = StrConv(abCustname(), vbUnicode)
lblAddress = StrConv(abAddress(), vbUnicode)
lblCity = StrConv(abCity(), vbUnicode)
lblState = StrConv(abState(), vbUnicode)
lblPhone = StrConv(abPhone(), vbUnicode)
lblStatus = StrConv(abStatus(), vbUnicode)

```

```

Loop

```

Examples: Calling an iSeries stored procedure by using Visual Basic: The Visual Basic programming examples listed below show a stored procedure call being prepared. Two statements are shown:

1. A statement for the creation of the stored procedure
2. A statement to prepare the call

Create the stored procedure only once. The definition that it provides is available to ODBC applications, as well as to integrated OS/400 applications.

Tips: Running and calling iSeries stored procedures:

Running a stored procedure on the iSeries server:

ODBC provides a standard interface for calling stored procedures. The implementation of stored procedures differs significantly across various databases. This simple example follows the recommended approach for running a stored procedure on the iSeries server:

1. Set up a **create procedure** statement for the stored procedure and create it. The creation of the stored procedure only needs to be done once it does not have to be done through ODBC. The definition that it provides is available to all ODBC as well as integrated OS/400 applications. This step can also help performance, as the Optimizer knows in advance the data type, the direction of the parameters, and the language of the procedure.
2. Prepare the stored procedure call.
3. Bind the parameters of the procedure, indicating whether each parameter is to be used for input to the procedure, output from the procedure, or input/output.
4. Call the stored procedure.

Calling iSeries stored procedures using Visual Basic:

Use care in coding the **SQLBindParameter** functions. Never use Visual Basic strings as a buffer when binding either columns (**SQLBindCol**) or parameters (**SQLBindParameter**). Instead, use byte arrays, which—unlike strings—will not be moved around in memory. See “Example: Using arrays of byte” for more information.

Pay careful attention to the data types that are involved. There may be subtle differences with those that you use with, for instance, a select statement. Also, ensure that you have an adequately sized buffer for output and input/output parameters. The way that you code the stored procedure on the iSeries server can affect performance significantly. Whenever possible, avoid closing the program with **exit()** in C language and with **SETON LR** in RPG language. Preferably, use **RETRN** or **return**, but you may need to re-initialize variables on each call, and by-pass file opens.

Example: Using arrays of byte: Because of the way Visual Basic stores and manages the String data type, using an array of Byte data type instead of a String variable is recommended for the following parameter types:

- Input/output parameters
- Output parameters
- Any parameter that contains binary data (rather than standard ANSI characters)
- Any input parameter that has a variable address which is set once, but referred to many times

The last case would be true for the if the application made multiple calls to **SQLExecute**, while modifying **Parm1** between each call. The following Visual Basic functions assist in converting strings and arrays of byte:

```
Public Sub Byte2String(InByte() As Byte, OutString As String)
    'Convert array of byte to string
    OutString = StrConv(InByte(), vbUnicode)
End Sub
```

```
Public Function String2Byte(InString As String, OutByte() As Byte) As Boolean
    'vb byte-array / string coercion assumes Unicode string
    'so must convert String to Byte one character at a time
    'or by direct memory access
    'This function assumes Lower Bound of array is 0
```

```
Dim I As Integer
Dim SizeOutByte As Integer
Dim SizeInString As Integer
```

```
SizeOutByte = UBound(OutByte) + 1
SizeInString = Len(InString)
```

```

'Verify sizes if desired

'Convert the string
For I = 0 To SizeInString - 1
    OutByte(I) = AscB(Mid(InString, I + 1, 1))
Next I
'If size byte array > len of string pad with Nulls for szString
If SizeOutByte > SizeInString Then      'Pad with Nulls
    For I = SizeInString To UBound(OutByte)
        OutByte(I) = 0
    Next I
End If

String2Byte = True
End Function

Public Sub ViewByteArray(Data() As Byte, Title As String)
'Display message box showing hex values of byte array

Dim S As String
Dim I As Integer
On Error GoTo VBANext

S = "Length: " & Str(UBound(Data) - LBound(Data) + 1) & " Data (in hex):"
For I = LBound(Data) To UBound(Data)
    If (I Mod 8) = 0 Then
        S = S & " "      'add extra space every 8th byte
    End If
    S = S & Hex(Data(I)) & " "
VBANext:
Next I
MsgBox S, , Title

End Sub

```

Example: Calling CL command stored procedures

It is possible to run iSeries server commands by using stored procedures. Simply call **Execute Command (QCMDExC)** to run the command. The process is relatively simple, but ensure that you include all of the zeros in the length parameter. Use the Remote Command API as an alternative.

The two examples that are provided here apply to ODBC programs. The first example enables the powerful SQL tracing facility that writes data into the joblog for the job running the SQL (in this case, the OS/400 server job).

The second example overcomes a restriction in SQL: its limited ability to work with multi-member files. You cannot create a multi-member file through CREATE TABLE. However, the following example shows you how to access with ODBC anything but the first member of a file that is created through DDS:

```

| Dim hStmt          As Long
|
| rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
| If rc <> SQL_SUCCESS Then
|     Call DspSQLError(SQL_HANDLE_DBC, ghDbc, "Problem: Allocating Debug Statement Handle")
| End If
|
| ' Note that the string within single quotes 'STRDBG UPDPROD(*YES)' is exactly 20 bytes
| cmd = "call qsys.qcmdexc('STRDBG UPDPROD(*YES)',0000000020.00000)"
|
| ' Put the iSeries job in debug mode
| rc = SQLExecDirect(hStmt, cmd, SQL_NTS)

```

```

| If rc <> SQL_SUCCESS Then
|   Call DspSQLError(SQL_HANDLE_STMT, hStmt, "Problem: Start Debug")
| End If
|
| rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, ovrhstmt)
| If rc <> SQL_SUCCESS Then
|   Call DspSQLError(SQL_HANDLE_DBC, ghDbc, "Problem: Allocating Override Statement Handle")
| End If
|
| ' Note that the string within single quotes 'OVRDBF FILE(BRANCH)... OVRSCOPE(*JOB)'
|   is exactly 68 bytes
|   cmd = "call qsys.qcmdexc('OVRDBF FILE(BRANCH) TOFILE(HOALIB/BRANCH) MBR(FRANCE)
|                                     OVRSCOPE(*JOB)',0000000068.00000)"
|
| ' Override the iSeries file to point to the 'france' member
| rc = SQLExecDirect(hStmt, cmd, SQL_NTS)
| If rc <> SQL_SUCCESS Then
|   Call DspSQLError(SQL_HANDLE_STMT, hStmt, "File Override")
| End If

```

Choosing an interface to access the ODBC driver

There are different programming interfaces that can be used with the iSeries Access for Windows ODBC Driver. Each interface has its strengths and weaknesses. Three of the more common programming interfaces are ActiveX Data Objects (ADO), Rapid Application Development (RAD) tools, and ODBC APIs. The supported languages, reasons for using, and sources of more information for these three interfaces, are provided below.

ActiveX Data Objects (ADO)

ADO refers to ActiveX Data Objects and is Microsoft's high level object model for data access.

- Supported programming languages:
 - Visual Basic
 - Active Server Pages (ASP)
 - Delphi
 - Visual Basic Script
 - any other language or script that supports ActiveX or COM
- Reasons to use this method:
 - Eliminates the coding of ODBC APIs
 - Supports switching providers, when needed
- Where to go for more information:
 - More on how to use ADO, see the ADO documentation that comes in MDAC:
<http://www.microsoft.com/data/doc.htm>
 - More on using the iSeries Access OLE-DB Provider through ADO refer to: "iSeries Access for Windows OLE DB Provider" on page 555
- Special notes:
 - To use ODBC through ADO an application needs to specify the MSDASQL provider in a connection string. MSDASQL converts ADO calls into ODBC API calls which communicate with the ODBC driver.
 - An example using an ADO connection string follows:

```
ConnectionString = "Provider=MSDASQL;Data Source=MYODBCDS;"
```

Rapid Application Development (RAD) tools

Rapid Application Development tools are tools that help in creating applications quickly. The tools make it so that the application writer does not have to know much about the ODBC specification.

- Supported programming languages:
 - Depends on which RAD tool is used.
 - Some of the more commonly used tools include Powerbuilder, Delphi, and Seagate Crystal Reports.
- Reasons to use this method:
 - Eliminates the coding of ODBC APIs
 - Works with multiple ODBC drivers using one program, with few or no changes
- Where to go for more information:
 - Refer to the documentation included with the RAD tool.


Direct ODBC API calls

Direct ODBC API calls are when an application is written directly to the ODBC specification.

- Supported programming language:
 - C/C++
- Reasons to use this method:
 - Allows direct control over which ODBC APIs are called so can be faster than using ADO objects or RAD tools
 - Designed to take advantage of driver-specific features
- Where to go for more information:
 - For information on the ODBC specification and some samples see the ODBC documentation that comes in MDAC: <http://www.microsoft.com/data/doc.htm>.
 - For more information about driver-specific features see “Implementation issues of ODBC APIs” on page 586

ODBC programming examples

For some examples on how to write ODBC applications see the links below under ODBC partial programming examples. For complete discussions and programming samples, refer to the following locations:

- To access ODBC programming samples (Visual Basic, C++, and Lotus Script programming environments), link to the IBM ftp site  on the Web. Select **index.txt** to see what programming examples are available and to download to your PC).
- For information on Stored Procedures and examples on how to call them see “Stored procedures” on page 632.
- Search for ODBC samples in Microsoft’s MSDN library or ODBC webpage. Examples can be found for Visual Basic, ADO, and C/C++.
- The C programming example in the Programmer’s Toolkit

ODBC partial programming examples:

The following ODBC programming examples demonstrate simple queries, and accessing and returning data by calling stored procedures. C/C++, Visual Basic and RPG programming language versions are provided. Note that many of the C/C++ samples are not complete programs.

- “Example: Visual C++ - Accessing and returning data by calling a stored procedure” on page 641
- “Example: Visual Basic - Accessing and returning data by calling a stored procedure” on page 642
- “Examples: RPG - Host code for ODBC stored procedures” on page 644
- “Using large objects (LOBs) and DataLinks with iSeries Access for Windows ODBC” on page 569

Example: Visual C++ - Accessing and returning data by calling a stored procedure

Only the code relevant to the stored procedure call has been included here. This code assumes the connection has already been established. See “Examples: RPG - Host code for ODBC stored procedures” on page 644 for the source code for the stored procedure.

Creating the stored procedure

```
/* Drop the old Procedure
strcpy(szDropProc,"drop procedure apilib.partqry2");

rc = SQLExecDirect(m_hstmt, (unsigned char *)szDropProc, SQL_NTS);

// This statement is used to create a stored procedure
// Unless the
// procedure is destroyed, this statement need never be re-created
strcpy(szCreateProc,"CREATE PROCEDURE APILIB.PARTQRY2 (INOUT P1 INTEGER," );
strcat(szCreateProc,"INOUT P2 INTEGER");
strcat(szCreateProc,"EXTERNAL NAME APILIB.SPROC2 LANGUAGE RPG GENERAL")

/* Create the new Procedure
rc = SQLExecDirect(m_hstmt, (unsigned char *)szCreateProc, SQL_NTS);
if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) {
    DspSQLError(m_henv, m_hdbc, SQL_NULL_HSTMT);
    return APIS_INIT_ERROR;
}
if(rc != SQL_SUCCESS) {
    DspSQLError(m_henv, m_hdbc, SQL_NULL_HSTMT);
    return APIS_INIT_ERROR;
}
}
```

Preparing the statements

```
// Prepare the procedure call
strcpy(szStoredProc, "call partqry2(?, ?)");
// Prepare the stored procedure statement
rc = SQLPrepare(m_hstmt, (unsigned char *) szStoredProc, strlen(szStoredProc));
if(rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) {
    DspSQLError(m_henv, m_hdbc, m_hstmt);
    return APIS_INIT_ERROR;
}
}
```

Binding the parameters

```
// Bind the parameters for the stored procedure

rc = SQLBindParameter(m_hstmt, 1, SQL_PARAM_INPUT_OUTPUT, SQL_C_LONG,
    SQL_INTEGER, sizeof(m_lOption), 0, &m_lOption, sizeof(m_lOption), &lcbon,
    &lcbOption);
rc |= SQLBindParameter(m_hstmt, 2, SQL_PARAM_INPUT_OUTPUT, SQL_C_LONG,
    SQL_INTEGER, sizeof(m_lPartNo), 0, &m_lPartNo, sizeof(m_lPartNo), &lcbon,
    &lcbOption);

// Bind the Columns
rc = SQLBindCol(m_hstmt, 1, SQL_C_SLONG, &m_lSPartNo,
    sizeof(m_lSPartNo), &lcbBuffer);
rc |= SQLBindCol(m_hstmt, 2, SQL_C_CHAR, &m_szSPartDesc,
    26, &lcbBuffer);
rc |= SQLBindCol(m_hstmt, 3, SQL_C_SLONG, &m_lSPartQty,
    sizeof(m_lSPartQty), &lcbBuffer);
```

```

rc |= SQLBindCol(m_hstmt, 4, SQL_C_DOUBLE, &m_dSPartPrice,
    sizeof(m_dSPartPrice), &lcbBuffer);
rc |= SQLBindCol(m_hstmt, 5, SQL_C_DATE, &m_dsPartDate,
    10, &lcbBuffer);

```

Calling the stored procedure

```

// Request a single record
m_lOption = ONE_RECORD;
m_lPartNo = PartNo;

// Run the stored procedure
rc = SQLExecute(m_hstmt);
if (rc != SQL_SUCCESS) {
    DspSQLError(m_henv, m_hdbc, m_hstmt);
    return APIS_SEND_ERROR;
}

// (Try to) fetch a record
rc = SQLFetch(m_hstmt);
if (rc == SQL_NO_DATA_FOUND) {
    // Close the cursor for repeated processing
    rc = SQLCloseCursor(m_hstmt);
    return APIS_PART_NOT_FOUND;
}
else if (rc != SQL_SUCCESS) {
    DspSQLError(m_henv, m_hdbc, m_hstmt);
    return APIS_RECEIVE_ERROR;
}

// If we are still here we have some data, so map it back
// Format and display the data
.
.
.

```

Example: Visual Basic - Accessing and returning data by calling a stored procedure

Visual Basic is able to call external functions that are found in DLLs. Since all ODBC drivers are DLLs, Visual Basic can be used to code directly to the ODBC APIs. By coding directly to the ODBC APIs a Visual Basic application can call an iSeries server stored procedure and return result values. See "Coding directly to ODBC APIs" on page 568 for more information. See "Examples: RPG - Host code for ODBC stored procedures" on page 644 for the source code for the stored procedure.

Creating the stored procedure

```

' This statement will drop an existing stored procedure
szDropProc = "drop procedure apilib.partqry2"

'* This statement is used to create a stored procedure
'* Unless the
'* procedure is destroyed, this statement need never be re-created
szCreateProc = "CREATE PROCEDURE APILIB.PARTQRY2 (INOUT P1 INTEGER,"
szCreateProc = szCreateProc & "INOUT P2 INTEGER)"
szCreateProc = szCreateProc & "EXTERNAL NAME APILIB.SPROC2 LANGUAGE RPG GENERAL"

'* Allocate statement handle
rc = SQLAllocHandle(SQL_HANDLE_STMT, ghDbc, hStmt)
If rc <> SQL_SUCCESS Then

```



```

        Call DisplayError(rc, "SQLAllocStmt failed.")
        Call DspSQLError(henv, SQL_NULL_HDBC, SQL_NULL_HSTMT)
    End If
    '* Drop the old Procedure
    rc = SQLExecDirect(hstmt, szDropProc, SQL_NTS)

    ' Create the new Procedure
    rc = SQLExecDirect(hstmt, szCreateProc, SQL_NTS)
    If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then
        Call DisplayError(rc, "SQLCreate failed.")
        Call DspSQLError(henv, hdbc, hstmt)
    End If

```

Preparing the statements

```

    '* This statement will be used to call the stored procedure
    szStoredProc = "call partqry2(?, ?)"
    '* Prepare the stored procedure call statement

    rc = SQLPrepare(hstmt, szStoredProc, Len(szStoredProc))
    If rc <> SQL_SUCCESS And rc <> SQL_SUCCESS_WITH_INFO Then
        Call DisplayError(rc, "SQLPrepare failed.")
        Call DspSQLError(henv, hdbc, hstmt)
    End If

```

Binding the parameters

```

    'Bind the parameters for the stored procedure
    rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG, _
        SQL_INTEGER, 1Len1, 0, sFlag, 1Len1, 1CbValue)

    If rc <> SQL_SUCCESS Then
        Call DisplayError(rc, "Problem binding parameter ")
    End If

    rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_SLONG, _
        SQL_INTEGER, 4, 0, 1PartNumber, 1Len2, 1CbValue)

    If rc <> SQL_SUCCESS Then
        Call DisplayError(rc, "Problem binding parameter ")
    End If

```

Calling the stored procedure

```

    rc = SQLExecute(hstmt)
    If !rc <> SQL_SUCCESS Then
        ' Free the statement handle for repeated processing
        rc = SQLFreeHandle(
            Call DspSQLError(henv, hdbc, hstmt)
    End If
    rc = SQLFetch(hstmt)
    If rc = SQL_NO_DATA_FOUND Then
        mnuClear_Click          'Clear screen
        txtPartNumber = 1PartNumber 'Show the part number not found
        Call DisplayMessage("RECORD NOT FOUND")
        .
        .
    Else
        'Get Description
        rc = SQLGetData(hstmt, 2, SQL_C_CHAR, sDescription, _
            25, 1CbBuffer)
    End If

```

```

'Get Quantity. SQLGetLongData uses alias SQLGetData
rc = SQLGetLongData(hstmt, 3, SQL_C_SLONG, lSQuantity, _
                    Len(lSQuantity), lcbBuffer)
'Get Price. SQLGetDoubleData uses alias SQLGetData
rc = SQLGetDoubleData(hstmt, 4, SQL_C_DOUBLE, dSPrice, _
                      Len(dSPrice), lcbBuffer)
'Get Received date
rc = SQLGetData(hstmt, 5, SQL_C_CHAR, sSReceivedDate, _
                10, lcbBuffer)
txtDescription = sSDescription 'Show description
txtQuantity = lSQuantity      'Show quantity
txtPrice = Format(dSPrice, "currency") 'Convert dSPrice to
txtReceivedDate = CDate(sSReceivedDate) 'Convert string to d
Call DisplayMessage("Record found")
End If

```

Examples: RPG - Host code for ODBC stored procedures

This program, **SPROC2**, is called from the client as a stored procedure via ODBC. It returns data to the client from the PARTS database file.

RPG/400® (non-ILE) example:

```

* THIS EXAMPLE IS WRITTEN IN RPG/400 (NON-ILE)
*
* DEFINES PART AS AN INTEGER (BINARY 4.0)
*
I#OPTDS      DS
I                                     B  1  40#OPT
I#PRTDS      DS
I                                     B  1  40#PART
C           *ENTRY      PLIST
C           PARM          #OPTDS
C           PARM          #PRTDS
* COPY PART NUMBER TO RPG NATIVE VARIABLE WITH SAME
* ATTRIBUTES OF FIELD IN PARTS MASTER FILE (PACKED DECIMAL 5,0)
C           Z-ADD#PART    PART    50
C           #OPT          CASEQ1    ONEREC
C           #OPT          CASEQ2    ALLREC
C           ENDCS
C           SETON          LR
C           RETRN
*
*****
C           ONEREC      BEGSR
*****
* PROCESS REQUEST FOR A SINGLE RECORD.
C/EXEC SQL DECLARE C1 CURSOR FOR
C+  SELECT
C+  PARTNO,
C+  PARTDS,
C+  PARTQY,
C+  PARTPR,
C+  PARTDT
C+
C+  FROM PARTS          -- FROM PART MASTER FILE
C+
C+  WHERE PARTNO = :PART
C+
C+  FOR FETCH ONLY      -- READ ONLY CURSOR
C/END-EXEC

```

```

C*
C/EXEC SQL
C+ OPEN C1
C/END-EXEC
C*
C/EXEC SQL
C+ SET RESULT SETS CURSOR C1
C/END-EXEC
C                                ENDSR
*****
C                                ALLREC  BEGSR
*****
* PROCESS REQUEST TO RETURN ALL RECORDS
C/EXEC SQL DECLARE C2 CURSOR FOR
C+ SELECT
C+ PARTNO,
C+ PARTDS,
C+ PARTQY,
C+ PARTPR,
C+ PARTDT
C+
C+ FROM PARTS          -- FROM PART MASTER FILE
C+
C+ ORDER BY PARTNO    -- SORT BY PARTNO
C+
C+ FOR FETCH ONLY    -- READ ONLY CURSOR
C/END-EXEC
C*
C/EXEC SQL
C+ OPEN C2
C/END-EXEC
C*
C/EXEC SQL
C+ SET RESULT SETS CURSOR C2
C/END-EXEC
C                                ENDSR

```

ILE-RPG example:

```

* This example is written in ILE-RPG
*
* Define option and part as integer
D#opt          s          10i 0
D#part         s          10i 0
* Define part as packed 5/0
Dpart          s          5p 0

C  *entry      plist
C              parm          #opt
C  part        parm          #part

C  #opt        caseq         1          onerec
C  #opt        caseq         2          allrec
C              endcs

C              eval          *inlr = *on
C              return

*
*****
C  onerec      begsr
*****
* Process request for a single record.

```

```

C/EXEC SQL DECLARE C1 CURSOR FOR
C+  SELECT
C+  PARTNO,
C+  PARTDS,
C+  PARTQY,
C+  PARTPR,
C+  PARTDT
C+
C+  FROM PARTS          -- FROM PART MASTER FILE
C+
C+  WHERE PARTNO = :PART
C+
C+
C+  FOR FETCH ONLY      -- READ ONLY CURSOR
C/END-EXEC
C*
C/EXEC SQL
C+  OPEN C1
C/END-EXEC
C*
C/EXEC SQL
C+  SET RESULT SETS CURSOR C1
C/END-EXEC
C
                endsr
*****
C  allrec      begsr
*****
* Process request to return all records
C/EXEC SQL DECLARE C2 CURSOR FOR
C+  SELECT
C+  PARTNO,
C+  PARTDS,
C+  PARTQY,
C+  PARTPR,
C+  PARTDT
C+
C+  FROM PARTS          -- FROM PART MASTER FILE
C+
C+
C+  ORDER BY PARTNO    -- SORT BY PARTNO
C+
C+  FOR FETCH ONLY     -- READ ONLY CURSOR
C/END-EXEC
C*
C/EXEC SQL
C+  OPEN C2
C/END-EXEC
C*
C/EXEC SQL
C+  SET RESULT SETS CURSOR C2
C/END-EXEC
C
                endsr

```

iSeries Access for Windows database APIs

iSeries Access for Windows database APIs provide a superset of the function that is provided in the ODBC interface. All of the ODBC function is provided, along with extensions that allow an application developer to take advantage of unique iSeries server functions. The iSeries Access for Windows database APIs provide access to iSeries database files through a call-level interface.

iSeries Access for Windows database APIs required files:

Header file	Import library	Dynamic Link Library
cwbdb.h	cwbapi.lib	cwbdb.dll

Programmer's Toolkit:

The Programmer's Toolkit provides Database documentation, access to the cwbdb.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select **Database** → **C/C++ APIs**.

iSeries Access for Windows database APIs topics:

- “iSeries Access for Windows database APIs overview”
- “Typical use of iSeries Access for Windows database APIs” on page 649
- “Objects that process data on the PC or iSeries server” on page 650
- “Code page support in Windows” on page 651
- Database APIs listing**
- “Example: Using SQL to access database functions” on page 845
- “Database APIs return codes” on page 22

Related topics:

- “iSeries system name formats for ODBC Connection APIs” on page 12
- “OEM, ANSI, and Unicode considerations” on page 12

iSeries Access for Windows database APIs overview

Use iSeries Access for Windows database SQL APIs to access database functions on the iSeries server. These functions can be grouped into three categories:

- Catalog information
- SQL functions
- Native database (NDB) functions

The iSeries Access for Windows Database APIs are built on an object-oriented base. Handles are used to provide an application access to the following classes of objects:

- “Connection object”
- “Catalog request object”
- “Native database (NDB) request object” on page 648
- “SQL request” on page 648
- “Data format object” on page 648
- “Parameter marker format object” on page 648
- “Data object” on page 649

Connection object

This class of object represents an iSeries database server module. The connection class is used to control the processing of the iSeries database server. This class gives the application control over such server attributes as naming convention and sort sequence. Connection objects are independent of each other. This means that it is possible to have connections to multiple iSeries servers or multiple connections to the same iSeries server. Each connection could have a unique set of server attributes.

All functional requests must be processed by a server. Therefore, an object of this class must be created before an application can create objects of other classes. When objects of other classes are created, the handle that represents a connection object is used to identify which connection (database server) will be used to service any functional requests for that object. This means that the server must be started (using the cwdbDB_StartServer call) before the function can be performed.

Catalog request object

This class of object is used to retrieve information about database and other SQL objects (SQL packages) from the iSeries server. Information that pertains to the following is available through the catalog request:

- Fields
- Files
- Foreign keys
- Indices

- Libraries
- Members
- Primary keys
- Relational databases (RDBs)
- Record formats
- SQL packages
- Statements that are stored in SQL packages
- Special columns

By using the catalog request, an application can control both the type of information that is to be returned and the objects for which the information is to be returned. For example, you can use a catalog request to return the name and description of all files whose names start with the letter Q in the QIWS library.

See “Catalog request APIs” on page 649 for more information.

Native database (NDB) request object

This class of object is used to manipulate database file objects on the iSeries server. This includes member manipulation (add, clear, remove) as well as creating and duplicating database files. In addition, by using **NDB requests** in association with SQL requests, an application can access data in members other than the first member of a file using SQL as the access method.

See “Native Database (NDB) request APIs” on page 650 for more information.

SQL request

This class of object is used to request SQL operations to be performed on the iSeries server.

The **SQL request** object allows an application to set various parameters that control the processing of SQL statements on the iSeries server. Among these parameters are the library and SQL package name that allows the application to use “extended dynamic” SQL. When extended dynamic SQL is used, SQL statements only need to be prepared once. The prepared statement is stored in the specified package and can be reused at a later time.

See “SQL request APIs” on page 650 for more information.

Data format object

This class of object describes data that is contained in a result set (for example, the result of a **select** statement or the result of a catalog request).

The **data format** contains a description for each item in the result set. That description includes: data type, data length, precision, scale, CCSID, and column name.

Since NDB requests do not return data, this class is not used in conjunction with NDB requests.

See “Objects that process data on the PC or iSeries server” on page 650 for more information.

Parameter marker format object

This class of objects describes data that corresponds to parameter markers that are contained in SQL statements.

The **parameter marker format** contains a description for each parameter marker in a prepared SQL statement. That description includes: data type, data length, precision, scale, and CCSID.

See “Objects that process data on the PC or iSeries server” on page 650 for more information.

Data object

This class is used to return result data to the calling application. Using a **data object** removes the responsibility from the calling application to create buffers large enough to contain result data. The data object itself manages how much storage is needed to contain the data.

See “Objects that process data on the PC or iSeries server” on page 650 for more information.

Typical use of iSeries Access for Windows database APIs

A connection object is required in order to perform any functional requests with iSeries Access for Windows database APIs. You first must create a connection handle. Once this is done, that handle can be used to override the default set of attributes of the server job such as naming convention (LIB/FILE vs. USER.TABLE), or to override the default sort sequence (by using the `cwbDB_SetNLSS` API). The server job then can be started by using the `cwbDB_StartServer` API.

Once the connection is created and the server is started, requests and other related objects can be created and processed. The iSeries server will not return any data until it is requested (by using one of the **cwbDB_Return*** APIs—see “`cwbDB_ReturnData`” on page 765). This, along with the ability to store parameters for a request on the iSeries server, allows an application to perform processing in an asynchronous manner.

When an application stores parameters for a request on the iSeries server, storage is allocated to contain result information for any operations that are performed on that request. This result information is saved until another operation is performed for that request. As a result, an application can create any number of requests and store the parameters for those requests. Operations that take longer to complete (like creating SQL collections) can be requested, and the application can continue its work on the PC, while the iSeries server is processing the request. When the application is ready to check the results of the operations, it then can request whatever information is appropriate for the request (SQLCA, host error information, data, and so on). Link to the following topics for descriptions of the three types of requests and a list of their corresponding APIs:

- “Catalog request APIs”
- “Native Database (NDB) request APIs” on page 650
- “SQL request APIs” on page 650

Catalog request APIs

The catalog request APIs consist of a group of APIs that allow an application to specify what object for which information is being requested. For example, if the application needs information that pertains to members of a database file, the following APIs likely would be called:

cwbDB_SetLibraryName

This qualifies the library for which the information will be retrieved. This may contain wildcard values (QIWS*) or special values such as *LIBL, *USRLIBL, and so on.

cwbDB_SetFileName

This qualifies the file for which the member information will be retrieved. This may contain wildcard characters.

cwbDB_SetMemberName*

This is optional and may contain wild-card characters.

cwbDB_ReturnData

This is required for data to be sent to the PC from the iSeries server. If this API is not used, the data will be kept on the iSeries server until it is requested, or until the next operation is performed.

cwbDB_RetrieveMemberInformation

One of the parameters on this API is a bitmap that indicates what information is to be returned. The following information can be returned for members:

- Library name
- File name

- Member name
- Member description

Once the information has been retrieved, the application will use the data format and the data object to process the data that has been retrieved. See “Objects that process data on the PC or iSeries server” for more information.

Native Database (NDB) request APIs

Native database (NDB) requests are used to manipulate database objects on the iSeries server. For example, the override database function can be used in conjunction with an SQL request to allow SQL to access members other than the first member in a file. The following NDB APIs would be used to accomplish this:

cwbDB_SetFileName

This is the file (table) that will be used in the SQL statement.

cwbDB_SetOverrideInformation*

This will indicate the file and member to be accessed.

cwbDB_ReturnHostErrorInfo

Since no data is available to be returned, this will provide status as to the result of the override request.

cwbDB_OverrideFile

This actually does the override request.

SQL request APIs

SQL requests are used to perform SQL operations on the iSeries server. There are some operations for which a combined function API is provided. For example, an application could call APIs to perform the following functions: prepare a statement, describe the result set, open a cursor by using the prepared statement, and fetch data from that cursor. Using the combined function API, the application would make one API call to perform all four of those functions (cwbDB_PrepareDescribeOpenFetch). The following APIs would be used to open a cursor and retrieve some data:

cwbDB_SetCursorName

This is the name of the cursor that will be used when fetching the data.

cwbDB_SetStatementName

This is the name that is used when referring to prepared SQL statements.

cwbDB_SetStatementText

This is the actual SQL statement that is to be prepared.

cwbDB_ReturnData

No data is returned unless it is requested by the application. For this example, we will request the data.

cwbDB_PrepareDescribeOpenFetch

This will process the statement and return data to the PC.

Objects that process data on the PC or iSeries server

There are three classes that are used by the application for processing data that is returned to the PC or for providing data to be processed by the iSeries server. These classes are:

Data format

The data format is used to describe data that is to be returned to the PC. It contains a description of each of the columns of data in the result set. This description includes the column name, length, and type. If the type of data is character data, the Coded Character Set Identifier (CCSID) is included. For numeric data, the description includes precision and scale. This information is used by the application to parse the data that is returned to the PC.

Parameter marker format

Parameter marker formats are similar to the data formats in that they describe data that is contained in a buffer. The difference is that the parameter marker format is used to describe data that the application is using as input to an SQL request. The information in the parameter marker format is used by Database APIs to parse through a buffer that contains data that is to be used to provide data values to an SQL statement.

Data object

The data object is a very simple object. It provides a pointer and length to data that is returned to the PC. As mentioned previously, using the data object provides a mechanism for the application to receive data without having to allocate storage of sufficient size to contain the data. That storage management is contained within the data object.

Code page support in Windows

In Windows, data can be manipulated in ASCII (OEM) or ANSI code pages. The default behavior of these iSeries Access for Windows database APIs is to use the ASCII code page. If you want your program to use the ANSI code page instead, use the `cwbNL_GetANSIcodePage` API to retrieve the ANSI code page, convert the code page to a CCSID with “`cwbNL_CodePageToCCSID`” on page 239, and then use `cwbDB_SetClientDataCCSID` and `cwbDB_SetClientHostErrorCCSID` to change the behavior of these APIs.

Note: Unicode is not supported by these APIs.

iSeries Access for Windows database APIs listing

The following iSeries Access for Windows database APIs are listed alphabetically, by function:

Function	iSeries Access for Windows database APIs
Database server attributes	cwbDB_ApplyAttributes cwbDB_CreateConnectionHandle cwbDB_CreateConnectionHandleEx cwbDB_DeleteConnectionHandle cwbDB_GetCommitmentControl cwbDB_GetDateFormat cwbDB_GetDateSeparator cwbDB_GetDecimalSeparator cwbDB_GetIgnoreDecimalDataError cwbDB_GetNamingConvention cwbDB_GetRelationalDBName cwbDB_GetServerFunctionalLevel cwbDB_GetTimeFormat cwbDB_GetTimeSeparator cwbDB_SetAllowAddStatementToPackage cwbDB_SetAmbiguousSelectOption cwbDB_SetAutoCommit cwbDB_SetCommitmentControl cwbDB_SetDateFormat cwbDB_SetDateSeparator cwbDB_SetDecimalSeparator cwbDB_SetDefaultSQLLibraryName cwbDB_SetIgnoreDecimalDataError cwbDB_SetLOBFieldThreshold cwbDB_SetNLSS cwbDB_SetNamingConvention cwbDB_SetRelationalDBName cwbDB_SetTimeFormat cwbDB_SetTimeSeparator cwbDB_StartServer cwbDB_StartServerDetailed cwbDB_StopServer

Function	iSeries Access for Windows database APIs
Catalog request	cwbdb_CreateCatalogRequestHandle cwbdb_DeleteCatalogRequestHandle cwbdb_RetrieveFieldInformation cwbdb_RetrieveFileInformation cwbdb_RetrieveForeignKeyInformation cwbdb_RetrieveIndexInformation cwbdb_RetrieveLibraryInformation cwbdb_RetrieveMemberInformation cwbdb_RetrievePackageStatementInformation cwbdb_RetrievePrimaryKeyInformation cwbdb_RetrieveRDBInformation cwbdb_RetrieveRecordFormatInformation cwbdb_RetrieveSpecialColumnInformation cwbdb_RetrieveSQLPackageInformation cwbdb_SetFieldName cwbdb_SetFileAttributes cwbdb_SetFileInfoOrdering cwbdb_SetFileType cwbdb_SetForeignKeyFileName cwbdb_SetForeignKeyLibName cwbdb_SetFormatName cwbdb_SetIndexType cwbdb_SetLongFileName cwbdb_SetMemberName cwbdb_SetNullable cwbdb_SetPackageName cwbdb_SetPrimaryKeyFileName cwbdb_SetPrimaryKeyLibName cwbdb_SetRDBName cwbdb_SetStatementType
Native database (NDB) request	cwbdb_AddLibraryToList cwbdb_AddMember cwbdb_ClearMember cwbdb_CreateDuplicateFile cwbdb_CreateNDBRequestHandle cwbdb_CreateSourcePhysicalFile cwbdb_DeleteFile cwbdb_DeleteNDBRequestHandle cwbdb_OverrideFile cwbdb_RemoveMember cwbdb_RemoveOverride cwbdb_SetAddLibraryName cwbdb_SetAddLibraryPosition cwbdb_SetAuthority cwbdb_SetBaseFile cwbdb_SetFileText cwbdb_SetMaximumMembers cwbdb_SetMemberText cwbdb_SetOverrideInformation cwbdb_SetRecordLength

Function	iSeries Access for Windows database APIs
SQL request	cwbDB_GetBaseColumnName cwbDB_GetBaseSchemaName cwbDB_GetBaseTableName cwbDB_ClearPackage cwbDB_Close cwbDB_Commit cwbDB_Connect cwbDB_CreatePackage cwbDB_CreateSQLRequestHandle cwbDB_DeletePackage cwbDB_DeleteSQLRequestHandle cwbDB_Describe cwbDB_DescribeParameterMarkers cwbDB_DynamicStreamFetch cwbDB_EndStreamFetch cwbDB_Execute cwbDB_ExecuteImmediate cwbDB_ExtendedDynamicStreamFetch cwbDB_Fetch cwbDB_GetExtendedColumnInfo cwbDB_MoreStreamData cwbDB_Open cwbDB_OpenDescribeFetch cwbDB_Prepare cwbDB_PrepareDescribe cwbDB_PrepareDescribeOpenFetch cwbDB_RetrieveLOBData cwbDB_ReturnExtendedDataFormat cwbDB_ReturnParameterMarkerFormat cwbDB_ReturnSQLCA cwbDB_Rollback cwbDB_SetBlockCount cwbDB_SetCursorName cwbDB_SetCursorReuse cwbDB_SetDescribeOption cwbDB_SetExtendedDataFormat cwbDB_SetFetchScrollOptions cwbDB_SetHoldIndicator cwbDB_SetParameterMarkerBlock cwbDB_SetParameterMarkers cwbDB_SetPrepareOption cwbDB_SetScrollableCursor cwbDB_SetStatementName cwbDB_SetStatementText cwbDB_SetStreamFetchSyncCount
Multiple requests types	cwbDB_GetData - Catalog, NDB, SQL cwbDB_ReturnData - Catalog, NDB, SQL cwbDB_ReturnDataFormat - Catalog, SQL cwbDB_ReturnHostErrorInfo - Catalog, NDB, SQL cwbDB_SetClientDataCCSID - Catalog, NDB, SQL cwbDB_SetClientHostErrorCCSID - Catalog, NDB, SQL cwbDB_SetClientInputCCSID - Catalog, NDB, SQL cwbDB_SetCursorReuse cwbDB_SetFileName - Catalog, NDB cwbDB_SetLibraryName - Catalog, NDB, SQL cwbDB_SetQueryTimeoutValue cwbDB_StoreRequestParameters - Catalog, NDB, SQL cwbDB_SetStaticCursorResultSetThreshold cwbDB_WriteLOBData

Function	iSeries Access for Windows database APIs
Data-description manipulation	cwbDB_CreateDataFormatHandle cwbDB_CreateDataHandle cwbDB_CreateParameterMarkerFormatHandle cwbDB_DeleteDataFormatHandle cwbDB_DeleteDataHandle cwbDB_DeleteParameterMarkerFormatHandle cwbDB_GetColumnCCSID cwbDB_GetColumnCount cwbDB_GetColumnLength cwbDB_GetColumnName cwbDB_GetColumnPrecision cwbDB_GetColumnScale cwbDB_GetColumnType cwbDB_GetConversionIndicator cwbDB_GetDataLength cwbDB_GetDataPointer cwbDB_GetLOBLocator cwbDB_GetLOBMaxSize cwbDB_GetParameterCCSID cwbDB_GetParameterCount cwbDB_GetParameterDirection cwbDB_GetParameterLength cwbDB_GetParameterName cwbDB_GetParameterPrecision cwbDB_GetParameterScale cwbDB_GetParameterType cwbDB_GetRowSize cwbDB_GetSizeOfParameters cwbDB_GetSizeOfInputParameters cwbDB_GetSizeOfOutputParameters cwbDB_IsParameterInput cwbDB_IsParameterInputOutput cwbDB_SetClientColumnToNumeric cwbDB_SetClientColumnToString cwbDB_SetClientParameterToNumeric cwbDB_SetClientParameterToString cwbDB_SetConversionIndicator cwbDB_SetConvert65535

cwbDB_AddLibraryToList

Purpose: Add a library to the iSeries server library list.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_AddLibraryToList(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_AddLibraryToList** API may be called after setting the position in the library list at which the library is to be added using the **cwbDB_SetAddLibraryPosition** API. The **cwbDB_AddLibraryToList** API should be called after setting the library name in the request via the **cwbDB_SetAddLibraryName** API. This API will result in a request datastream flowing to the iSeries server and, if requested, a response to the request flowing back to the client. A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_AddMember

Purpose: Add a member to a file on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_AddMember(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_AddMember** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order to determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_ApplyAttributes

Purpose: Activates the changes that were made to server attributes by previous calls - (naming convention, commitment control, etc.) Use this to change server attributes after the server has been started.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ApplyAttributes(  
    cwbDB_ConnectionHandle connection,  
    cwbSV_ErrHandle        errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle API**. The messages may be retrieved through the **cwbSV_GetErrText API**. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: This API is only needed if the server attributes are changed after the server has been started (**cwbDB_StartServer**).

| **cwbDB_GetBaseColumnName**

| **Purpose:** Returns the base column name, if it exists, for a column of data.

| **Syntax:**

```
|  
| unsigned int CWB_ENTRY cwbDB_GetBaseColumnName(  
|                                     cwbDB_FormatHandle format,  
|                                     unsigned long      columnPosition  
|                                     cwbDB_DataHandle   columnHandle  
|                                     cwbSV_ErrHandle   errorHandle);
```

| **Parameters:**

| **cwbDB_FormatHandle format - input**

| Handle to a data format object.

| **unsigned long columnPosition - input**

| Specifies the relative position of the column.

| **cwbDB_DataHandle columnHandle - input**

| Handle to a data object which will contain the base column name.

| **cwbSV_ErrHandle errorHandle - input**

| Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle API**. The messages may be retrieved through the **cwbSV_GetErrText API**. If the parameter is set to zero, no messages will be retrievable.

| **Return Codes:** The following list shows common return values.

| **CWB_OK**

| Successful completion.

| **CWB_INVALID_API_HANDLE**

| Invalid request handle.

| **Usage:**

cwbDB_ClearMember

Purpose: Clear data from a member in an iSeries server file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ClearMember(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_ClearMember** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_GetBaseSchemaName

Purpose: Returns the base schema name, if it exists, for a column of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetBaseSchemaName(  
    cwbDB_FormatHandle format,  
    unsigned long      columnPosition  
    cwbDB_DataHandle  schemaHandle  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long columnPosition - input

Specifies the relative position of the column.

cwbDB_DataHandle schemaHandle - input

Handle to a data object which will contain the extended schema name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage:

| **cwbDB_GetBaseTableName**

| **Purpose:** Returns the base table name, if it exists, for a column of data.

| **Syntax:**

```
|  
| unsigned int CWB_ENTRY cwbDB_GetBaseTableName(  
|                               cwbDB_FormatHandle  format,  
|                               unsigned long         columnPosition  
|                               cwbDB_DataHandle     tableHandle  
|                               cwbSV_ErrHandle     errorHandle);
```

| **Parameters:**

| **cwbDB_FormatHandle format - input**

| Handle to a data format object.

| **unsigned long columnPosition - input**

| Specifies the relative position of the column.

| **cwbDB_DataHandle tableHandle - input**

| Handle to a data object which will contain the base table name.

| **cwbSV_ErrHandle errorHandle - input**

| Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

| **Return Codes:** The following list shows common return values.

| **CWB_OK**

| Successful completion.

| **CWB_INVALID_API_HANDLE**

| Invalid request handle.

| **Usage:**

cwbDB_ClearPackage

Purpose: Clear all statements from an SQL package.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ClearPackage(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_ClearPackage** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_Close

Purpose: Close an open cursor.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Close(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Close** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_Commit

Purpose: Perform a commit operation to commit a unit of work.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Commit(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Commit** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_Connect

Purpose: Perform a Distributed Relational Database Architecture™ (DRDA) connection management function. This API is used to establish and switch between connections to other Relational Databases.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Connect(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Connect** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to either **cwbDB_ReturnHostErrorInfo** or **cwbDB_ReturnSQLCA** prior to this call will allow an application to determine the success of the API operation.

cwbDB_CreateCatalogRequestHandle

Purpose: Allocate a handle to a database request. This handle will be used on subsequent API calls that request object information.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateCatalogRequestHandle(  
    cwbDB_ConnectionHandle connection,  
    cwbDB_RequestHandle *request,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to the connection which will be used when servicing the request.

cwbDB_RequestHandle *request - output

Pointer to a **cwbDB_RequestHandle** where the handle of the Request will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: None

cwbDB_CreateConnectionHandle

Purpose: Allocate a handle to an iSeries database access server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateConnectionHandle(  
    char *systemName,  
    cwbDB_ConnectionHandle *connection,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

char *systemName - input

Pointer to an ASCIIZ string that contains the name of the server from which database requests will be serviced.

cwbDB_ConnectionHandle *connection - output

Pointer to a **cwbDB_ConnectionHandle** where the handle of the connection will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: None

cwbDB_CreateConnectionHandleEx

Purpose: Allocate a handle to an iSeries database access server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateConnectionHandleEx(  
    cwbCO_SysHandle sysHandle,  
    cwbDB_ConnectionHandle* connection,  
    cwbSV_ErrHandle errorHandle );
```

Parameters:

cwbCO_SysHandle sysHandle - input

Handle to a server object.

cwbDB_ConnectionHandle *connection - output

Pointer to a **cwbDB_ConnectionHandle** where the handle of the connection will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful Completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: This function requires that you previously have issued **cwbCO_CreateSystem**.

cwbDB_CreateDataFormatHandle

Purpose: Allocate a handle to a description of SQL data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateDataFormatHandle(  
    cwbDB_ConnectionHandle connection,  
    cwbDB_FormatHandle *format,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

cwbDB_FormatHandle *format - output

Pointer to a **cwbDB_FormatHandle** where the handle of the data format will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_CreateDataHandle

Purpose: Allocate a handle to a data object.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateDataHandle(  
    cwbDB_DataHandle *dataHandle,  
    cwbSV_ErrHandle  errorHandler);
```

Parameters:

cwbDB_DataHandle *dataHandle - output

Pointer to a **cwbDB_DataHandle** where the handle of a data object will be returned.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

Usage: The **cwbDB_CreateDataHandle** is used prior to requesting various pieces of information to be returned to an application. In general, if the information being requested has a varying length, the information will be returned using a data handle. This mechanism moves the responsibility of allocating the memory that is to contain the data from the calling application to the API. When finished with the data handle, the **cwbDB_DeleteDataHandle** API should be called to free any resources that are associated with the data handle.

cwbDB_CreateDuplicateFile

Purpose: Create a file based on existing file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateDuplicateFile(  
    cwbDB_RequestHandle request,  
    cwb_Boolean copyDataIndicator,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwb_Boolean copyDataIndicator - input

Boolean value that indicates whether the data from the base file is to be copied into the duplicate file.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use one of the defined values for the copyDataIndicator:

CWBDB_DO_NOT_COPY_DATA

CWBDB_COPY_DATA

This API is not valid for List or SQL requests. The **cwbDB_CreateDuplicateFile** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_CreateNDBRequestHandle

Purpose: Allocate a handle to a database request. This handle will be used on subsequent API calls that request operations to be performed with iSeries file objects.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateNDBRequestHandle(  
    cwbDB_ConnectionHandle connection,  
    cwbDB_RequestHandle *request,  
    cwbSV_ErrHandle      errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to the connection which will be used when servicing the request.

cwbDB_RequestHandle *request - output

Pointer to a cwbDB_RequestHandle where the handle of the Request will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: None

cwbDB_CreatePackage

Purpose: Create an SQL package for preparing statements.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreatePackage(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_CreatePackage** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_CreateParameterMarkerFormatHandle

Purpose: Allocate a handle to a description of SQL parameter marker data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateParameterMarkerFormatHandle(  
    cwbDB_ConnectionHandle connection,  
    cwbDB_FormatHandle *format,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

cwbDB_FormatHandle *format - output

Pointer to a **cwbDB_FormatHandle** where the handle of the parameter marker format will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_CreateSourcePhysicalFile

Purpose: Create a source file on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateSourcePhysicalFile(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_CreateSourcePhysicalFile** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_CreateSQLRequestHandle

Purpose: Allocate a handle to a database request. This handle will be used on subsequent API calls that request SQL services.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_CreateSQLRequestHandle(  
    cwbDB_ConnectionHandle connection,  
    cwbDB_RequestHandle *request,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to the connection which will be used when servicing the request.

cwbDB_RequestHandle *request - output

Pointer to a **cwbDB_RequestHandle** where the handle of the Request will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_NOT_ENOUGH_MEMORY

Insufficient memory.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: None

cwbDB_DeleteCatalogRequestHandle

Purpose: Deallocates a request handle.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteCatalogRequestHandle(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_DeleteConnectionHandle

Purpose: Deallocates the handle to the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteConnectionHandle(  
    cwbDB_ConnectionHandle connection,  
    cwbSV_ErrHandle        errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: None

cwbDB_DeleteDataFormatHandle

Purpose: Deallocates a format handle.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteDataFormatHandle(  
    cwbDB_FormatHandle format,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_DeleteDataHandle

Purpose: Deallocates a data handle.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteDataHandle(  
    cwbDB_DataHandle dataHandle,  
    cwbSV_ErrHandle  errorHandle);
```

Parameters:

cwbDB_DataHandle dataHandle - input

Handle to a data object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_DeleteFile

Purpose: Delete a file from an iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteFile(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_DeleteFile** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_DeleteNDBRequestHandle

Purpose: Deallocates a request handle.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteNDBRequestHandle(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_DeletePackage

Purpose: Delete an SQL package.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeletePackage(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_DeletePackage** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_DeleteParameterMarkerFormatHandle

Purpose: Deallocates a format handle.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteParameterMarkerFormatHandle(  
    cwbDB_FormatHandle format,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a parameter marker format object.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_DeleteSQLRequestHandle

Purpose: Deallocates a request handle.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DeleteSQLRequestHandle(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_Describe

Purpose: Describes a prepared statement. If there is no result set, no column descriptions will be returned.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Describe(  
                                cwbDB_RequestHandle request,  
                                cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Describe** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnDataFormat** is needed in order get the description of the data. Calling **cwbDB_ReturnDataFormat** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_DescribeParameterMarkers

Purpose: Describes the parameter markers for a prepared statement. If the statement is an "UPDATE WHERE CURRENT OF CURSOR", the cursor must be open before the describe parameter markers can be performed.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DescribeParameterMarkers(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Describe** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnParameterMarkerFormat** is needed in order get the description of the parameter markers. Calling **cwbDB_ReturnParameterMarkerFormat** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_DynamicStreamFetch

Purpose: This API will prepare a select statement, open a cursor and fetch all resulting data. The row data will be returned to the application in blocks, the size of which will be optimized for the communication mechanism. To get additional blocks, use the **cwbDB_MoreStreamData** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_DynamicStreamFetch(  
    cwbDB_RequestHandle request,  
    char *statementText,  
    cwbDB_DataHandle data,  
    cwbDB_DataHandle indicators,  
    cwbDB_FormatHandle formatHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *statementText - input

Pointer to an ASCII string containing select text.

cwbDB_DataHandle data - input

Handle to a data object into which the returned data will be placed.

cwbDB_DataHandle indicators - input

Handle to a data object into which the returned data indicators will be placed. There is one indicator value for each column value of each row of data that is returned from the iSeries server. The indicator will be a negative number if the value for the column is NULL. If an error occurs while converting the data, a character 'E' will be placed in that column's indicator field.

cwbDB_FormatHandle formatHandle - input

Handle to a data format that contains a description of the returned data.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_EndStreamFetch

Purpose: Cancel the stream fetch operation before all the data has been returned.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_EndStreamFetch(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_Execute

Purpose: Execute a prepared SQL statement.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Execute(  
                                cwbDB_RequestHandle request,  
                                cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Execute** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_ExecutImmediate

Purpose: Prepare and execute an SQL statement.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ExecutImmediate(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_ExecutImmediate** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_ExtendedDynamicStreamFetch

Purpose: This API will perform a stream fetch (see previous API) for a statement that is already prepared in an SQL package.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ExtendedDynamicStreamFetch(  
    cwbDB_RequestHandle request,  
    char                *libraryName,  
    char                *packageName,  
    char                *statementName,  
    cwbDB_DataHandle   data,  
    cwbDB_DataHandle   indicators,  
    cwbDB_FormatHandle formatHandle,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *libraryName - input

Pointer to an ASCIIZ string containing library name.

char *packageName - input

Pointer to an ASCIIZ string containing package name.

char *statementName - input

Pointer to an ASCIIZ string containing statement name.

cwbDB_DataHandle data - input

Handle to a data object into which the returned data will be placed.

cwbDB_DataHandle indicators - input

Handle to a data object into which the returned data indicators will be placed. There is one indicator value for each column value of each row of data that is returned from the iSeries server. The indicator will be a negative number if the value for the column is NULL. If an error occurs while converting the data, a character 'E' will be placed in that column's indicator field.

cwbDB_FormatHandle formatHandle - input

Handle to a data format that contains a description of the returned data.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_Fetch

Purpose: Fetch a row or block of rows (this is controlled by the **cwbDB_SetBlockCount** API) from an open cursor.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Fetch(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Fetch** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client. Please note that fetched data will not be returned unless the data is requested (using the **cwbDB_ReturnData** API).

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_GetColumnCCSID

Purpose: Returns the Coded Character Set Identifier (CCSID) for a specified column of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetColumnCCSID(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      columnPosition,  
    unsigned short     *dataCCSID,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long columnPosition - input

Specifies the relative position of the column.

unsigned short *dataCCSID - output

Pointer to a short integer to contain the CCSID for the specified column.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetColumnCount

Purpose: Returns the number of columns of data that are described by the data format.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetColumnCount(  
    cwbDB_FormatHandle format,  
    unsigned long *columnCount,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long *columnCount - output

Pointer to an unsigned long integer which will contain the column count.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_GetColumnLength

Purpose: Returns the length (in bytes) of the data for a specified column.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetColumnLength(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      columnPosition,  
    unsigned long      *dataLength,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long columnPosition - input

Specifies the relative position of the column.

unsigned long *dataLength - output

Pointer to short integer to contain the data length.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetColumnName

Purpose: Returns the column name (if it exists) for a column of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetColumnName(  
    cwbDB_FormatHandle format,  
    unsigned long      columnPosition,  
    cwbDB_DataHandle  columnHandle,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long columnPosition - input

Specifies the relative position of the column.

cwbDB_DataHandle columnHandle - input

handle to a data object which will contain the column name

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_GetColumnPrecision

Purpose: Returns the precision for a specified column of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetColumnPrecision(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      columnPosition,  
    unsigned short     *dataPrecision,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long columnPosition - input

Specifies the relative position of the column.

unsigned short *dataPrecision - output

Pointer to short integer to contain the data precision.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetColumnScale

Purpose: Returns the scale for a specified column of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetColumnScale(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      columnPosition,  
    unsigned short     *dataScale,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long columnPosition - input

Specifies the relative position of the column.

unsigned short *dataScale - output

Pointer to short integer to contain the data scale.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetColumnType

Purpose: Returns the data type for a specified column of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetColumnType(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      columnPosition,  
    signed short       *dataType,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long columnPosition - input

Specifies the relative position of the column.

signed short *dataType - output

Short integer which will contain the data type.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

If the server information is requested, the type returned is the SQL type. If the local information is requested, see the defined values:

CWBDB_PCNOCONVERSION

CWBDB_PCSTRING

CWBDB_PCLONG

CWBDB_PCSHORT

CWBDB_PCFLOAT

CWBDB_PCDOUBLE

CWBDB_PCPACKED

CWBDB_PCZONED

CWBDB_PCINVALIDTYPE

CWBDB_PCVARSTRING

CWBDB_PCGRAPHIC

CWBDB_PCVARGRAPHIC

cwbDB_GetCommitmentControl

Purpose: Get the current commitment control level.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetCommitmentControl(  
    cwbDB_ConnectionHandle connection,  
    unsigned short *commitmentLevel,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short *commitmentLevel - output

Pointer to an unsigned short where the current value will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value that is returned will be one of the following:

CWBDB_NONE

CWBDB_CURSOR_STABILITY

CWBDB_CHANGE

CWBDB_ALL

cwbDB_GetConversionIndicator

Purpose: Gets the indicator that says whether data is to be converted between the client and host format.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetConversionIndicator(  
    cwbDB_FormatHandle format,  
    cwb_Boolean *conversionIndicator,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

cwb_Boolean *conversionIndicator - output

CWB_FALSE indicates that no conversion CWB_TRUE indicates conversion

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_GetData

Purpose: Get the requested data from the host. This data can include the selected data, data format, host return code, and SQLCA.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetData(  
                                cwbDB_RequestHandle request,  
                                cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_GetData** API should be called after requesting the desired data (using the **cwbDB_Return*** APIs). This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_GetDataLength

Purpose: Returns the length of the data contained in a data object.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetDataLength(  
    cwbDB_DataHandle dataHandle,  
    unsigned long *dataLength,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_DataHandle dataHandle - input

Handle to a data object.

unsigned long *dataLength - output

Unsigned long integer to contain the length of the data.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_GetDataPointer

Purpose: Returns the address of the data in a data object.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetDataPointer(  
    cwbDB_DataHandle dataHandle,  
    char **data,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_DataHandle dataHandle - input

Handle to a data object.

char **data - output

Pointer to pointer to the data buffer.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_GetDateFormat

Purpose: Get the current date format. See **cwbDB_SetDateFormat** for additional information about date formats.

Syntax:

```
unsigned int CWB_ENTRY  cwbDB_GetDateFormat(  
                        cwbDB_ConnectionHandle connection,  
                        unsigned short      *dateFormat,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short *dateFormat - output

Pointer to an unsigned short where the current date format value will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

CWBDB_INVALID_ARG_API

Value specified is not in range.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value that is returned will be one of the following:

Format name	Date format constant	Value
-----	-----	-----
Julian	CWBDB_DATE_FMT_JUL	0
month day year	CWBDB_DATE_FMT_MDY	1
day month year	CWBDB_DATE_FMT_DMY	2
year month day	CWBDB_DATE_FMT_YMD	3
USA	CWBDB_DATE_FMT_USA	4
ISO	CWBDB_DATE_FMT_ISO	5
IBM Japan	CWBDB_DATE_FMT_JIS	6
IBM Europe	CWBDB_DATE_FMT_EUR	7

cwbDB_GetDateSeparator

Purpose: Get the current date separator. See **cwbDB_SetDateSeparator** for additional information about date separators.

Syntax:

```
unsigned int CWB_ENTRY  cwbDB_GetDateSeparator(  
                        cwbDB_ConnectionHandle connection,  
                        unsigned short      *dateSeparator,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short *dateSeparator - output

Pointer to an unsigned short where the current date data separator value will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value that is returned will be one of the following:

Date separator	Date separator constant
Slash	CWBDB_DATE_SEP_SLASH
Dash	CWBDB_DATE_SEP_DASH
Period	CWBDB_DATE_SEP_PERIOD
Comma	CWBDB_DATE_SEP_COMMA
Blank	CWBDB_DATE_SEP_BLANK

cwbDB_GetDecimalSeparator

Purpose: Get the current decimal separator. See **cwbDB_SetDecimalSeparator** for additional information about decimal separators.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetDecimalSeparator(  
    cwbDB_ConnectionHandle connection,  
    unsigned short *decimalSeparator,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short *decimalSeparator - output

Pointer to an unsigned short where the current decimal separator value will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value that is returned will be one of the following:

Time separator	Time separator constant
-----	-----
Period	CWBDB_DECIMAL_SEP_PERIOD
Comma	CWBDB_DECIMAL_SEP_COMMA

| **cwbDB_GetExtendedColumnInfo**

| **Purpose:** Returns the fixed-length portion of the extended column information.

| **Syntax:**

```
|  
| unsigned int CWB_ENTRY cwbDB_GetExtendedColumnInfo(  
|         cwbDB_FormatHandle    format ,  
|         unsigned long         columnPosition  
|         unsigned long         *columnInfo  
|         cwbSV_ErrHandle       errorHandle);
```

| **Parameters:**

| **cwbDB_FormatHandle format - input**

| Handle to a data format object.

| **unsigned long columnPosition - input**

| Specifies the relative position of the column.

| **unsigned long *columnInfo - output**

| Pointer to 4-byte integer to contain the extended column information.

| **cwbSV_ErrHandle errorHandle - input**

| Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

| **Return Codes:** The following list shows common return values.

| **CWB_OK**

| Successful completion.

| **CWB_INVALID_API_HANDLE**

| Invalid request handle.

| **Usage:**

cwbDB_GetIgnoreDecimalDataError

Purpose: Get the current setting for the decimal data error indicator.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetIgnoreDecimalDataError(  
    cwbDB_ConnectionHandle connection,  
    unsigned short *ignoreDecimalError,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short *ignoreDecimalError - output

Pointer to an unsigned short where the current value will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value returned will be one of the following:

CWBDB_IGNORE_ERROR

CWBDB_CORRECT_ERROR

cwbDB_GetLabelName

Purpose: Returns the label name (if it exists) for a column of data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetLabelName(  
    cwbDB_FormatHandle    format,  
    unsigned long         columnPosition  
    cwbDB_DataHandle     labelHandle  
    cwbSV_ErrHandle      errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to an extended format object.

unsigned long columnPosition - input

Specifies the relative position of the column.

cwbDB_DataHandle labelHandle - input

Handle to a data object which will contain the label name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage:

cwbDB_GetLOBLocator

Purpose: Returns the LOB Locator for a specified parameter.

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

unsigned long *dataLocator - output

Pointer to a long integer to contain the locator for the specified parameter.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

cwbDB_GetLOBMaxSize

Purpose: Returns the LOB Maximum size for a specified parameter.

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long columnPosition - input

Specifies the relative position of the column

unsigned long *maxSize - output

Pointer to a long integer to contain the LOB maximum size for the specified parameter.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

cwbDB_GetNamingConvention

Purpose: Get the naming convention (SQL or native iSeries server) that is in effect for the specified connection.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetNamingConvention(  
    cwbDB_ConnectionHandle connection,  
    unsigned short *namingConvention,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short *namingConvention - output

Pointer to an unsigned short where the current naming convention will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value that is returned will be one of the following:

CWBDB_PERIOD_NAME_CONV

CWBDB_SLASH_NAME_CONV

cwbDB_GetParameterCCSID

Purpose: Returns the Coded Character Set Identifier (CCSID) for a specified parameter.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetParameterCCSID(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      parameterPosition,  
    unsigned short     *dataCCSID,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

unsigned short *dataCCSID - output

Pointer to a short integer to contain the CCSID for the specified parameter.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetParameterCount

Purpose: Returns the number of parameters that are described by the data format.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetParameterCount(  
    cwbDB_FormatHandle format,  
    unsigned long *parameterCount,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long *parameterCount - output

Pointer to an unsigned long integer which will contain the parameter count.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_GetParameterDirection

Purpose: Returns the parameter direction.

Parameters:

cwbDB_FormatHandle format - input

Handle to a parameter marker format object.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

unsigned short* columnDirection

Receives the column direction, which will be one of the following: CWBDB_PM_INPUT_ONLY, CWBDB_PM_INPUT_OUTPUT, or CWBDB_PM_OUTPUT_ONLY.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

cwbDB_GetParameterLength

Purpose: Returns the length (in bytes) of the data for a specified parameter.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetParameterLength(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      parameterPosition,  
    unsigned long      *dataLength,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

unsigned long *dataLength - output

Pointer to short integer to contain the data length.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetParameterName

Purpose: Returns the parameter name (if it exists) for a column of data.

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long parameterPosition - input

Specifies the relative position of the parameter

cwbDB_DataHandle parameterHandle - input

Handle to a data object which will contain the parameter name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

cwbDB_GetParameterPrecision

Purpose: Returns the precision for a specified parameter.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetParameterPrecision(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      parameterPosition,  
    unsigned short     *dataPrecision,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

unsigned short *dataPrecision - output

Pointer to short integer to contain the data precision.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetParameterScale

Purpose: Returns the scale for a specified parameter.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetParameterScale(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      parameterPosition,  
    unsigned short     *dataScale,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

unsigned short *dataScale - output

Pointer to short integer to contain the data scale.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetParameterType

Purpose: Returns the data type for a specified parameter.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetParameterType(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      parameterPosition,  
    signed short       *dataType,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

signed short *dataType - output

Short integer which will contain the data type.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

If the server information is requested, the type returned is the SQL type. If the local information is requested, see the defined values:

CWBDB_PCNOCONVERSION

CWBDB_PCSTRING

CWBDB_PCLONG

CWBDB_PCSHORT

CWBDB_PCFLOAT

CWBDB_PCDOUBLE

CWBDB_PCPACKED

CWBDB_PCZONED

CWBDB_PCINVALIDTYPE

CWBDB_PCVARSTRING

CWBDB_PCGRAPHIC

CWBDB_PCVARGRAPHIC

cwbDB_GetRelationalDBName

Purpose: Get the current relational database name (usually system or server name).

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

char * relationalDBName - output

Pointer to buffer 18 characters long to receive the database name (Not null terminated).

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The `cwbDB_ApplyAttributes` API must be called after `cwbDB_SetAllowAddStatementToPackage` in order for the new value to take affect.

| **cwbDB_SetRelationalDBName**

| **Purpose:** Set the current relational database name.

| **Parameters:**

| **cwbDB_ConnectionHandle connection - input**

| Handle to a request object.

| **char * relationalDBName - input**

| Pointer to an 18 character string containing the relational database name. A special value of *SYSBAS indicates that a connection should be made to *SYSBAS RDB. This value should be used if a connection to the server ASP (SYSBAS) RDB is desired. Note: This name should be blank padded to 18 characters.

| **cwbSV_ErrHandle errorHandler - input**

| Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

| **Return Codes:** The following list shows common return values.

| **CWB_OK**

| Successful completion.

| **CWB_INVALID_API_HANDLE**

| Invalid request handle.

| **CWBDB_FUNCTION_NOT_VALID_AFTER_CONNECT**

| Cannot change independent disk pool (independent ASP) after connected.

| **CWB_API_ERROR**

| General API failure.

| **Usage:** If a call to cwbDB_SetRelationalDBName has not been made then the default database is used. The RDB can only be set before connecting to the server. This call is used to switch to a specific independent disk pool (independent ASP) while connecting the server.

cwbDB_GetRowSize

Purpose: Returns the size (in bytes) of the data described by the data format.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetRowSize(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      *rowSize,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long *rowSize - output

Pointer to an unsigned long integer which will contain the row size.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data.

cwbDB_GetServerFunctionalLevel

Purpose: Get the current server functional level.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetServerFunctionalLevel(  
    cwbDB_ConnectionHandle connection,  
    char *serverFunctionalLevel,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

char * serverFunctionalLevel - output

Pointer to buffer 11 characters long to receive the server's functional level.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data.

cwbDB_GetSizeOfParameters

Purpose: Returns the size (in bytes) of the all data described by parameter marker format.

Parameters:

cwbDB_FormatHandle format - input

Handle to a parameter marker format object.

unsigned long *bufferSize - output

Pointer to an unsigned long integer which will contain the parameter buffer size.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

cwbDB_GetSizeOfInputParameters

Purpose: Returns the size (in bytes) of the input data described by parameter marker format.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetSizeOfInputParameters(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      *inputSize,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a parameter marker format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long *inputSize - output

Pointer to an unsigned long integer which will contain the row size.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetSizeOfOutputParameters

Purpose: Returns the size (in bytes) of the output data described by parameter marker format.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_GetSizeOfOutputParameters(  
    cwbDB_FormatHandle format,  
    unsigned long      location,  
    unsigned long      *inputSize,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a parameter marker format object.

unsigned long location - input

Indicates whether the server or local information is to be returned.

unsigned long *outputSize - output

Pointer to an unsigned long integer which will contain the row size.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the location parameter, use one of the defined values:

CWBDB_SYSTEM

CWBDB_LOCAL

cwbDB_GetTimeFormat

Purpose: Get the current time format. See **cwbDB_SetTimeFormat** for additional information about time formats.

Syntax:

```
unsigned int CWB_ENTRY  cwbDB_GetTimeFormat(  
                        cwbDB_ConnectionHandle connection,  
                        unsigned short      *timeFormat,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short *timeFormat - output

Pointer to an unsigned short where the current time format value will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value that is returned will be one of the following:

Format name	Time format constant
-----	-----
Hours minutes seconds	CWBDB_TIME_FMT_HMS
USA	CWBDB_TIME_FMT_USA
ISO	CWBDB_TIME_FMT_ISO
IBM Europe	CWBDB_TIME_FMT_EUR
IBM Japan	CWBDB_TIME_FMT_JIS

cwbDB_GetTimeSeparator

Purpose: Get the current time separator. See **cwbDB_SetTimeSeparator** for additional information about time separators.

Syntax:

```
unsigned int CWB_ENTRY  cwbDB_GetTimeSeparator(  
                        cwbDB_ConnectionHandle connection,  
                        unsigned short      *timeSeparator,  
                        cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short *timeSeparator - output

Pointer to an unsigned short where the current time separator value will be returned.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: The **cwbDB_StartServer** API must be called before this API can return valid data. The value that is returned will be one of the following:

Time separator	Time separator constant
Colon	CWBDB_TIME_SEP_COLON
Period	CWBDB_TIME_SEP_PERIOD
Comma	CWBDB_TIME_SEP_COMMA
Blank	CWBDB_TIME_SEP_BLANK

cwbDB_IsParameterInput

Purpose: Returns a Boolean value indicating whether the parameter is input only.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_IsParameterInput(  
    cwbDB_FormatHandle format,  
    unsigned long      parameterPosition,  
    cwb_Boolean        *parameterIsInput,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a parameter marker format object.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

cwb_Boolean *parameterIsInput - output

Pointer to a Boolean indicating if the parameter is input only.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_IsParameterInputOutput

Purpose: Returns a Boolean value indicating whether the parameter is input and output.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_IsParameterInputOutput(  
    cwbDB_FormatHandle  format,  
    unsigned long       parameterPosition,  
    cwb_Boolean         *parameterIsInputOutput,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a parameter marker format object.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

cwb_Boolean *parameterIsInputOutput - output

Pointer to a Boolean indicating if the parameter is input and output.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_MoreStreamData

Purpose: This API will get the next block of stream fetch data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_MoreStreamData(  
    cwbDB_RequestHandle request,  
    cwbDB_DataHandle data,  
    cwbDB_DataHandle indicators,  
    cwbDB_FormatHandle formatHandle,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbDB_DataHandle data - input

Handle to a data object into which the returned data will be placed.

cwbDB_DataHandle indicators - input

Handle to a data object into which the returned data indicators will be placed. There is one indicator value for each column value of each row of data that is returned from the iSeries server. The indicator will be a negative number if the value for the column is NULL. If an error occurs while converting the data, a character 'E' will be placed in that column's indicator field.

cwbDB_FormatHandle formatHandle - input

Handle to a data format that contains a description of the returned data.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_Open

Purpose: Open a cursor.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Open(  
                                cwbDB_RequestHandle request,  
                                unsigned char      openOptions,  
                                cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned char openOptions - input

Input value for open options indicator.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the openOptions, use the defined values

CWBDB_READ

CWBDB_WRITE

CWBDB_UPDATE

CWBDB_DELETE

CWBDB_OPEN_ALL - Provided for convenience

This API is not valid for NDB or catalog requests. The **cwbDB_Open** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_OpenDescribeFetch

Purpose: This API combines the open, describe and fetch operations. This combined function is valuable when the statement is already prepared (extended dynamic SQL).

Syntax:

```
unsigned int CWB_ENTRY cwbDB_OpenDescribeFetch(  
    cwbDB_RequestHandle request,  
    unsigned char       openOptions,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned char openOptions - input

Input value for open options indicator.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the openOptions, use the defined values:

CWBDB_READ

CWBDB_WRITE

CWBDB_UPDATE

CWBDB_DELETE

CWBDB_OPEN_ALL - Provided for convenience

This API is not valid for NDB or catalog requests. The **cwbDB_OpenDescribeFetch** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client. Please note that fetched data will not be returned unless the data is requested (using the **cwbDB_ReturnData** API).

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_OverrideFile

Purpose: Override database file reference to another file/member.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_OverrideFile(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_OverRideFile** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_Prepare

Purpose: Prepares an SQL statement. If an SQL package has been set, this API will prepare a statement into the package.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Prepare(  
                                cwbDB_RequestHandle request,  
                                cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Prepare** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

cwbDB_PrepareDescribe

Purpose: This API combines the prepare and describe operations. The advantage of using this API is that the SQL component is called only once.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_PrepareDescribe(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_PrepareDescribe** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnDataFormat** is needed in order get the description of the data. Calling **cwbDB_ReturnDataFormat** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_PrepareDescribeOpenFetch

Purpose: This API combines the prepare, describe, open, and fetch operations. By combining these operations, performance will improve because only one call is made to the SQL component on the host.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_PrepareDescribeOpenFetch(  
    cwbDB_RequestHandle request,  
    unsigned char       openOptions,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned char openOptions - input

Input value for open options indicator.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the **openOptions**, use the defined values

CWBDB_READ

CWBDB_WRITE

CWBDB_UPDATE

CWBDB_DELETE

CWBDB_OPEN_ALL - Provided for convenience

This API is not valid for NDB or catalog requests. The **cwbDB_PrepareDescribeOpenFetch** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client. Please note that fetched data will not be returned unless the data is requested (using the **cwbDB_ReturnData** API).

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RemoveMember

Purpose: Remove a member from an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RemoveMember(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_RemoveMember** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_RemoveOverride

Purpose: Remove an override from a file reference.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RemoveOverride(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_RemoveOverRide** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to **cwbDB_ReturnHostErrorInfo** is needed in order determine the success of the operation for this API. Calling **cwbDB_ReturnHostErrorInfo** prior to calling this API will result in a synchronous operation (the application will not get control back until the result is returned to the PC from the iSeries server).

cwbDB_RetrieveFieldInformation

Purpose: Get information about the fields in an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveFieldInformation(  
                cwbDB_RequestHandle request,  
                unsigned long      retrieveInformation,  
                cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the fields.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values

- CWBDB_GET_FLD_LIB
- CWBDB_GET_FLD_REMARKS
- CWBDB_GET_FLD_FILE
- CWBDB_GET_FLD_NAME
- CWBDB_GET_FLD_DESC
- CWBDB_GET_FLD_DATA_TYPE
- CWBDB_GET_FLD_LEN
- CWBDB_GET_FLD_NULL
- CWBDB_GET_FLD_RADIX
- CWBDB_GET_FLD_PREC
- CWBDB_GET_FLD_SCALE

```
rc = cwbDB_RetrieveFieldInformation( requestHandle,
```

```
CWBDB_GET_FLD_FILE |
```

```
CWBDB_GET_FLD_NAME |
```

```
CWBDB_GET_FLD_DATA_TYPE |
```

```
CWBDB_GET_FLD_PREC |
```

```
CWBDB_GET_FLD_SCALE,
```


errorHandle);

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveFieldInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveFileInformation

Purpose: Get information about files on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveFileInformation(  
                cwbDB_RequestHandle request,  
                unsigned long      retrieveInformation,  
                cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the files.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_FILE_LIB  
CWBDB_GET_FILE_REMARKS  
CWBDB_GET_FILE_NAME  
CWBDB_GET_FILE_ATTRIB  
CWBDB_GET_FILE_DESC  
CWBDB_GET_FILE_COL_CNT  
CWBDB_GET_FILE_AUTH
```

```
rc = cwbDB_RetrieveFileInformation( requestHandle,  
    CWBDB_GET_FILE_NAME |  
    CWBDB_GET_FILE_ATTRIB |  
    CWBDB_GET_FILE_DESC |  
    CWBDB_GET_FILE_COL_CNT |  
    CWBDB_GET_FILE_AUTH,  
    errorHandler );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveFileInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API

call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveForeignKeyInformation

Purpose: Get information about foreign keys for an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveForeignKeyInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for foreign keys.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

Foreign Key Primary Key Information constants

```
CWBDB_GET_FG_PRKEY_LIB  
CWBDB_GET_FG_PRKEY_FILE  
CWBDB_GET_FG_PRKEY_COL_ID
```

Foreign Key Information constants

```
CWBDB_GET_FG_KEY_LIB  
CWBDB_GET_FG_KEY_FILE  
CWBDB_GET_FG_KEY_COL_ID  
CWBDB_GET_FG_KEY_SEQ  
CWBDB_GET_FG_KEY_UPDATE  
CWBDB_GET_FG_KEY_DELETE
```

```
rc = cwbDB_RetrievePrimaryKeyInformation( requestHandle,  
    CWBDB_GET_FG_PRKEY_LIB |  
    CWBDB_GET_FG_PRKEY_FILE |  
    CWBDB_GET_FG_PRKEY_COL_ID |  
    CWBDB_GET_FG_KEY_LIB |  
    CWBDB_GET_FG_KEY_FILE |  
    CWBDB_GET_FG_KEY_COL_ID |  
    CWBDB_GET_FG_KEY_SEQ |
```

```
CWBDB_GET_FG_KEY_UPDATE |
CWBDB_GET_FG_KEY_DELETE, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveForeignKeyInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveIndexInformation

Purpose: Get information about the indices for an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveIndexInformation(  
                cwbDB_RequestHandle request,  
                unsigned long      retrieveInformation,  
                cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the indices.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_IDX_LIB  
CWBDB_GET_IDX_TBL_NAME  
CWBDB_GET_IDX_UNIQUE  
CWBDB_GET_IDX_IDX_LIB  
CWBDB_GET_IDX_IDX_NAME  
CWBDB_GET_IDX_COL_CNT  
CWBDB_GET_IDX_COL_NAME  
CWBDB_GET_IDX_COL_SEQ  
CWBDB_GET_IDX_COLLAT
```

```
rc = cwbDB_RetrieveIndexInformation( requestHandle,  
    CWBDB_GET_IDX_TBL_NAME |  
    CWBDB_GET_IDX_UNIQUE |  
    CWBDB_GET_IDX_IDX_LIB |  
    CWBDB_GET_IDX_IDX_NAME |  
    CWBDB_GET_IDX_COL_CNT, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveIndexInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and

subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveLibraryInformation

Purpose: Get information about a library or list of libraries.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveLibraryInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the libraries.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_LIBRARY_NAME  
CWBDB_GET_LIBRARY_DESC
```

```
rc = cwbDB_RetrieveLibraryInformation( requestHandle,  
    CWBDB_GET_LIBRARY_NAME |  
    CWBDB_GET_LIBRARY_DESC, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveLibraryInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveLOBData

Purpose: Retrieve LOB Data.

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbDB_DataHandle data - input

unsigned long locator - input

unsigned long size - input

unsigned long start - input

unsigned long columnIndex - input

Column Index one based column number. This is an optional parameter used to retrieve lob data for more than one row. Must be zero if not used.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

cwbDB_RetrieveMemberInformation

Purpose: Get information about members of an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveMemberInformation(  
                cwbDB_RequestHandle request,  
                unsigned long      retrieveInformation,  
                cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the members.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_MBR_LIB  
CWBDB_GET_MBR_FILE  
CWBDB_GET_MBR_NAME  
CWBDB_GET_MBR_DESC
```

```
rc = cwbDB_RetrieveMemberInformation( requestHandle,  
    CWBDB_GET_MBR_LIB |  
    CWBDB_GET_MBR_FILE |  
    CWBDB_GET_MBR_NAME |  
    CWBDB_GET_MBR_DESC, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveMemberInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrievePackageStatementInformation

Purpose: Get information about statements stored in an SQL package on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrievePackageStatementInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the SQL statements.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_SQLSTMT_LIB  
CWBDB_GET_SQLSTMT_PKG  
CWBDB_GET_SQLSTMT_NAME  
CWBDB_GET_SQLSTMT_TYPE  
CWBDB_GET_SQLSTMT_TEXT  
CWBDB_GET_SQLSTMT_PM_CNT
```

```
rc = cwbDB_RetrievePackageStatementInformation( requestHandle,  
    CWBDB_GET_SQLSTMT_NAME |  
    CWBDB_GET_SQLSTMT_TYPE |  
    CWBDB_GET_SQLSTMT_TEXT, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrievePackageStatementInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrievePrimaryKeyInformation

Purpose: Get information about primary keys for an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrievePrimaryKeyInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for primary keys.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_PR_KEY_LIB  
CWBDB_GET_PR_KEY_FILE  
CWBDB_GET_PR_KEY_COL_ID  
CWBDB_GET_PR_KEY_COL_SEQ
```

```
rc = cwbDB_RetrievePrimaryKeyInformation( requestHandle,  
    CWBDB_GET_PR_KEY_LIB |  
    CWBDB_GET_PR_KEY_FILE |  
    CWBDB_GET_PR_KEY_COL_ID |  
    CWBDB_GET_PR_KEY_COL_SEQ, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrievePrimaryKeyInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveRDBInformation

Purpose: Get information about a relational database on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveRDBInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the relational database.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_RDB_NAME  
CWBDB_GET_RDB_DEVICE  
CWBDB_GET_RDB_MODE  
CWBDB_GET_RDB_RMTLOC  
CWBDB_GET_RDB_LOCLCOC  
CWBDB_GET_RDB_RMTNET  
CWBDB_GET_RDB_TPNAME  
CWBDB_GET_RDB_DESC  
CWBDB_GET_RDB_TPNDISP  
CWBDB_GET_RDB_PGM  
CWBDB_GET_RDB_PGMLIB  
CWBDB_GET_RDB_PGMLEVEL
```

```
rc = cwbDB_RetrieveRDBInformation( requestHandle,  
    CWBDB_GET_RDB_NAME |  
    CWBDB_GET_RDB_RMTLOC |  
    CWBDB_GET_RDB_RMTNET |  
    CWBDB_GET_RDB_TPNAME |  
    CWBDB_GET_RDB_DESC, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveRDBInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveRecordFormatInformation

Purpose: Get information about the record formats for an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveRecordFormatInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the record formats.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_FMT_LIB  
CWBDB_GET_FMT_FILE  
CWBDB_GET_FMT_NAME  
CWBDB_GET_FMT_REC_LEN  
CWBDB_GET_FMT_DESC
```

```
rc = cwbDB_RetrieveRecordFormatInformation( requestHandle,  
    CWBDB_GET_FMT_LIB |  
    CWBDB_GET_FMT_FILE |  
    CWBDB_GET_FMT_NAME |  
    CWBDB_GET_FMT_REC_LEN |  
    CWBDB_GET_FMT_DESC, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveRecordFormatInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveSpecialColumnInformation

Purpose: Get information about special columns for an iSeries file.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveSpecialColumnInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the columns.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_SP_COL_LIB  
CWBDB_GET_SP_COL_TABLE  
CWBDB_GET_SP_COL_COL_NAME  
CWBDB_GET_SP_COL_DATA_TYPE  
CWBDB_GET_SP_COL_PRECISION  
CWBDB_GET_SP_COL_LENGTH  
CWBDB_GET_SP_COL_SCALE
```

```
rc = cwbDB_RetrieveSpecialColumnInformation( requestHandle,  
    CWBDB_GET_SP_COL_LIB |  
    CWBDB_GET_SP_COL_TABLE |  
    CWBDB_GET_SP_COL_COL_NAME |  
    CWBDB_GET_SP_COL_DATA_TYPE |  
    CWBDB_GET_SP_COL_PRECISION |  
    CWBDB_GET_SP_COL_LENGTH |  
    CWBDB_GET_SP_COL_SCALE, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveSpecialColumnInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and

subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_RetrieveSQLPackageInformation

Purpose: Get information about an SQL package on the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_RetrieveSQLPackageInformation(  
    cwbDB_RequestHandle request,  
    unsigned long      retrieveInformation,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long retrieveInformation - input

Bitmap that indicates what information is to be retrieved for the SQL packages.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use the defined values:

```
CWBDB_GET_SQLPKG_LIB  
CWBDB_GET_SQLPKG_NAME  
CWBDB_GET_SQLPKG_DESC
```

```
rc = cwbDB_RetrieveSQLPackageInformation( requestHandle,  
    CWBDB_GET_SQLPKG_LIB |  
    CWBDB_GET_SQLPKG_NAME |  
    CWBDB_GET_SQLPKG_DESC, errorHandle );
```

This API is not valid for NDB or SQL requests. The **cwbDB_RetrieveSQLPackageInformation** API should be called after setting the desired values in the request. This API will result in a request datastream flowing to the iSeries server and if requested, a response to the request flowing back to the client.

A call to the **cwbDB_ReturnData** API is needed prior to calling this API if the application is to process the data immediately. If the application is to operate asynchronously, then the call to **cwbDB_ReturnData** and subsequently **cwbDB_GetData** are needed after this API in order to get the data that result from this API call. Once the data is returned, information in the data format handle is used to determine how to parse the data.

cwbDB_ReturnData

Purpose: Instructs the API to return the data that is in the result set for the operation.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ReturnData(  
    cwbDB_RequestHandle request,  
    cwbDB_DataHandle data,  
    cwbDB_DataHandle indicators,  
    cwbDB_FormatHandle formatHandle,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbDB_DataHandle data - input

Handle for the data being returned. This address is returned when the data is received from the iSeries server on completion of a function request.

cwbDB_DataHandle indicators - input

Handle which will be used to return the address of the null value/error indicators being returned. There is one indicator value for each column value that is to be returned (for each column of each row) The indicator will be a negative number if the value for the column is NULL. If an error occurs while converting the data, a character 'E' will be placed in that columns indicator field. This address is returned when the data is received from the iSeries server on completion of a function request.

cwbDB_FormatHandle formatHandle - input

Handle to a data format that contains a description of the returned data.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_ReturnData** API is used to instruct the iSeries server to return the data which results from an operation (either an SQL fetch operation or a catalog retrieval operation). After calling this API, the next API call for the request that results in a datastream to flow to the server will result in the requested data being returned to the application.

cwbDB_ReturnDataFormat

Purpose: Instructs the API to return the format of the data to be returned.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ReturnDataFormat(  
    cwbDB_RequestHandle request,  
    cwbDB_FormatHandle formatHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbDB_FormatHandle formatHandle - input

Handle to a data format that contains a description of the returned data.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_ReturnDataFormat** API is used to instruct the iSeries server to return the data format which describes a set of selected data. After calling this API, the next API call for the request that results in a datastream to flow to the server will result in the requested data being returned to the application.

cwbDB_ReturnExtendedDataFormat

Purpose: Instructs the API to return the Extended version format of the data to be returned.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ReturnExtendedDataFormat(  
    cwbDB_RequestHandle request,  
    cwbDB_FormatHandle formatHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbDB_FormatHandle formatHandle - input

Handle to a data format that contains a description of the returned data, including the extended data.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_SERVER_FUNCTION_NOT_AVAILABLE

The host server is not at the required level to support this feature.

Usage: The **cwbDB_ReturnExtendedDataFormat** API is used to instruct the iSeries server to retrieve the extended data format information, in addition to the base data format information.

This API is used instead of the **cwbDB_ReturnDataFormat()** API when the extended data format information is required in addition to the base data format information.

The extended format data includes the information retrieved using the following APIs:

- **cwbDB_GetExtendedColumnInfo**
- **cwbDB_GetBaseColumnName**
- **cwbDB_GetBaseSchemaName**
- **cwbDB_GetBaseTableName**
- **cwbDB_GetLabelName**

After calling this API, the next API call for the request that results in a datastream to flow to the server will result in the requested data being returned to the application.

If the host server is not at the required level to support this feature, then the non-extended version of the data format will be returned, and subsequent calls to get extended data will return default values.

cwbDB_ReturnHostErrorInfo

Purpose: Instructs the API to return host error information when a function is performed on the host server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ReturnHostErrorInfo(  
    cwbDB_RequestHandle request,  
    unsigned short *hostErrorClass,  
    signed long *hostErrorCode,  
    cwbDB_DataHandle hostMsgID,  
    cwbDB_DataHandle firstLevelMessageText,  
    cwbDB_DataHandle secondLevelMessageText,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short *hostErrorClass - input

Pointer to location where the error class will be returned. This class indicates which database server module encountered an error.

- 0 - no error
- 1 - SQL functional error
- 2 - SQL parameter error
- 3 - List functional error
- 4 - List parameter error
- 5 - NDB functional error
- 6 - NDB parameter error
- 7 - General server error
- 8 - User exit error

signed long *hostErrorCode - input

Pointer to location where the return code from the server module will be placed.

cwbDB_DataHandle hostMsgID - input

Handle to a data object that will contain the host message identifier. If this parameter is set to 0, the host message identifier will not be retrieved.

cwbDB_DataHandle firstLevelMessageText - input

Handle to a data object that will contain the host first level message text. If this parameter is set to 0, the first level message text will not be retrieved.

cwbDB_DataHandle secondLevelMessageText - input

Handle to a data object that will contain the host second level message text. If this parameter is set to 0, the second level message text will not be retrieved.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The `cwbDB_ReturnHostErrorInfo` API is used to instruct the iSeries server to return the error or diagnostic information pertaining to a functional request. After calling this API, the next API call for the request that results in a datastream to flow to the server will result in the requested data being returned to the application.

cwbDB_ReturnParameterMarkerFormat

Purpose: Instructs the API to return the format of the parameter marker data for an SQL statement.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ReturnParameterMarkerFormat(  
    cwbDB_RequestHandle request,  
    cwbDB_FormatHandle formatHandle,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbDB_FormatHandle formatHandle - input

Handle to a parameter marker format that will contain the description of parameter data.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_ReturnParameterMarkerFormat** API is used to instruct the iSeries server to return the format which describes a set parameter markers for a prepared statement. After calling this API, the next API call for the request that results in a datastream to flow to the server will result in the requested data being returned to the application.

cwbDB_ReturnSQLCA

Purpose: Instructs the API to return the SQL Communication Area (SQLCA).

Syntax:

```
unsigned int CWB_ENTRY cwbDB_ReturnSQLCA(  
    cwbDB_RequestHandle request,  
    cwbDB_SQLCA *SQLca,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

struct cwbDB_SQLCA *SQLca - input

Pointer to a structure that will contain SQLCA returned from the host.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_ReturnSQLCA** API is used to instruct the iSeries server to return the SQL Communication Area (SQLCA). After calling this API, the next API call for the request that results in a datastream to flow to the server will result in the requested data being returned to the application.

cwbDB_Rollback

Purpose: Perform a rollback operation.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_Rollback(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests. The **cwbDB_Rollback** API should be called after setting the desired values in the request. This API results in a request datastream flowing to the iSeries server, and if requested, a response to the request flowing back to the client.

cwbDB_SetAddLibraryName

Purpose: Add a library to the iSeries library list.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetAddLibraryName(  
    cwbDB_RequestHandle request,  
    const char          *addLibraryName,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

const char *addLibraryName - input

The name of the library to be added to the library list.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_AddLibrary** API should be called after calling this API. The **cwbDB_SetAddLibraryPosition** API may be called before or after this API is called, but before **cwbDB_AddLibrary** is called. This API is not valid for List or SQL requests.

cwbDB_SetAddLibraryPosition

Purpose: Sets the position at which to add a library to the library list via the **cwbDB_AddLibraryToList** API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetAddLibraryPosition(  
    cwbDB_RequestHandle request,  
    const unsigned short position,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

const unsigned short position - input

The position in the library list to add the library name set via **cwbDB_SetAddLibraryName**. Use one of the following defined constants:

DB_ADD_LIBRARY_TO_FRONT - Add library to front of list

DB_ADD_LIBRARY_TO_END - Add library to end of list

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests. The **cwbDB_AddLibrary** API should be called after calling this API.

cwbDB_SetAllowAddStatementToPackage

Purpose: Sets server attribute for the connection to indicate if statements can be added to the package.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetAllowAddStatementToPackage(  
    cwbDB_ConnectionHandle connection,  
    cwb_Boolean allowAdd,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server..

cwb_Boolean allowAdd - input

Indicates whether SQL statements should be added to the package, if one is in use. CWB_FALSE indicates don't allow statements to be added. CWB_TRUE indicates add statement allowed. Default is allow add.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The cwbDB_ApplyAttributes API must be called after cwbDB_SetAllowAddStatementToPackage in order for the new value to take affect.

cwbDB_SetAmbiguousSelectOption

Purpose: Sets server attribute for the connection to indicate the explicit updateability.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetAmbiguousSelectOption(  
    cwbDB_ConnectionHandle connection,  
    unsigned short         updateability,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server..

unsigned short updateability - input

Indicates if SQL SELECT statements which do not have explicit FOR FETCH ONLY or FOR UPDATE OF clauses specified should be updateable or read-only. The default is updateable.

Use one of these two predefined values:

CWBDB_UPDATEABLE

CWBDB_READONLY

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The cwbDB_ApplyAttributes API must be called after cwbDB_SetAllowAddStatementToPackage in order for the new value to take affect.

cwbDB_SetAuthority

Purpose: Set the public authority for a file that will be created through the API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetAuthority(  
    cwbDB_RequestHandle request,  
    unsigned short      authority,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short authority - input

Long integer that indicates the public authority for a newly created file.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid authority value

Usage: Use one of the defined values:

CWBDB_SET_LIBRARY_CREATE_AUTHORITY

CWBDB_SET_ALL_AUTHORITY

CWBDB_SET_CHANGE_AUTHORITY

CWBDB_SET_EXCLUDE_AUTHORITY

CWBDB_SET_USE_AUTHORITY

CWBDB_SET_SAME_AUTHOR

This API is not valid for List or SQL requests.

cwbDB_SetAutoCommit

Purpose: Set an indicator that indicates if implicit commits will be done on the server.

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server..

unsigned short autoCommit - input

Indicates if auto commit will be done.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: Use one of the defined values:

CWBDB_AUTO_COMMIT

CWBDB_NO_AUTO_COMMIT

The default if not set is iplicit commits will be done.

The cwbDB_ApplyAttributes API must be called after cwbDB_SetAutoCommit in order for the new value to take affect.

cwbDB_SetBaseFile

Purpose: Set the name of a base file for creating a new file with the same format through the API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetBaseFile(  
    cwbDB_RequestHandle request,  
    char                *baseLibraryName,  
    char                *baseFileName,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *baseLibraryName - input

Pointer to an ASCIIZ string that contains the base library name to be used when creating a new file.

char *baseFileName - input

Pointer to an ASCIIZ string that contains the base file name to be used when creating a new file.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is used in preparation for **cwbDB_CreateDuplicateFile**. This API is not valid for List or SQL requests.

cwbDB_SetBlockCount

Purpose: Set the number of rows to be blocked together when fetching data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetBlockCount(  
    cwbDB_RequestHandle request,  
    unsigned long      blockCount,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long blockCount - input

Input value for block count.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_SetClientColumnToNumeric

Purpose: Sets the information for a column description for string data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetClientColumnToNumeric(  
    cwbDB_FormatHandle format,  
    unsigned long      columnPosition,  
    signed short      columnType,  
    unsigned long      columnLength,  
    unsigned short     columnPrecision,  
    unsigned short     columnScale,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long columnPosition - input

Specifies the relative position of the column.

signed short columnType - input

Specifies the numeric type to be used.

unsigned long columnLength - input

Only used if the type is zoned or packed decimal

unsigned short columnPrecision - input

Only used if the type is zoned or packed decimal

unsigned short columnScale - input

Only used if the type is zoned or packed decimal

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the **columnType** parameter, use one of the defined values:

- CWBDB_PCLONG
- CWBDB_PCSHORT
- CWBDB_PCFLOAT
- CWBDB_PCDOUBLE
- CWBDB_PCPACKED
- CWBDB_PCZONED

cwbDB_SetClientColumnToString

Purpose: Sets the information for a column description for string data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetClientColumnToString(  
    cwbDB_FormatHandle format,  
    unsigned long      columnPosition,  
    signed short       columnType,  
    unsigned long      columnLength,  
    unsigned short     columnCCSID,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long columnPosition - input

Specifies the relative position of the column.

signed short columnType - input

Specifies the string type to be used.

unsigned long columnLength - input

Specifies the column length to be used.

unsigned short columnCCSID - input

Specifies the column CCSID (Coded Character Set Identifier) to be used.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the **columnType** parameter, use one of the defined values:

CWBDB_PCSTRING

CWBDB_PCVARSTRING

CWBDB_PCGRAPHIC

CWBDB_PCVARGRAPHIC

cwbDB_SetClientDataCCSID

Purpose: Sets the CCSID (Coded Character Set ID) for the client. The new CCSID value will be used when converting EBCDIC data from the iSeries server. Use **cwbDB_SetClientHostErrorCCSID** to set the CCSID used when converting host error information.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetClientDataCCSID(  
    cwbDB_ConnectionHandle connection,  
    unsigned short         clientDataCCSID,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short clientCCSID - input

Specifies the CCSID (Coded Character Set Identifier) to be used.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be called any time after a connection handle has been created.

cwbDB_SetClientInputCCSID

Purpose: Sets the CCSID (Coded Character Set Identifier) for data being input such as file names, SQL statement text, and so on. The new CCSID value will be used when converting EBCDIC data from the iSeries server. Use **cwbDB_SetClientHostErrorCCSID** to set the CCSID used when converting host error information.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetClientInputCCSID(  
    cwbDB_ConnectionHandle connection,  
    unsigned short         inputCCSID,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short inputCCSID - input

Specifies the CCSID to be used.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be called any time after a connection handle has been created.

cwbDB_SetClientHostErrorCCSID

Purpose: Sets the CCSID (Coded Character Set ID) for the client. The new CCSID value will be used when converting EBCDIC server messages. Use **cwbDB_SetClientDataCCSID** to change the CCSID used for converting data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetClientHostErrorCCSID(  
    cwbDB_ConnectionHandle connection,  
    unsigned short         clientHostErrorCCSID,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short clientHostErrorCCSID - input

Specifies the CCSID (Coded Character Set Identifier) to be used.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be called any time after a connection handle has been created.

cwbDB_SetClientParameterToNumeric

Purpose: Sets the information for a parameter description for string data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetClientParameterToNumeric(  
    cwbDB_FormatHandle format,  
    unsigned long      parameterPosition,  
    signed short       parameterType,  
    unsigned long      parameterLength,  
    unsigned short     parameterPrecision,  
    unsigned short     parameterScale,  
    cwbSV_ErrHandle   errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

signed short parameterType - input

Specifies the numeric type to be used.

unsigned long parameterLength - input

Only used if the type is zoned or packed decimal

unsigned short parameterPrecision - input

Only used if the type is zoned or packed decimal

unsigned short parameterScale - input

Only used if the type is zoned or packed decimal

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the **parameterType** parameter, use one of the defined values:

- CWBDB_PCLONG
- CWBDB_PCSHORT
- CWBDB_PCFLOAT
- CWBDB_PCDOUBLE
- CWBDB_PCPACKED
- CWBDB_PCZONED

cwbDB_SetClientParameterToString

Purpose: Sets the information for a parameter description for string data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetClientParameterToString(  
    cwbDB_FormatHandle format,  
    unsigned long      parameterPosition,  
    signed short       parameterType,  
    unsigned long      parameterLength,  
    unsigned short     parameterCCSID,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

unsigned long parameterPosition - input

Specifies the relative position of the parameter.

signed short parameterType - input

Specifies the string type to be used.

unsigned long parameterLength - input

Specifies the parameter length to be used.

unsigned short parameterCCSID - input

Specifies the parameter CCSID (Coded Character Set Identifier) to be used.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For the **parameterType** parameter, use one of the defined values:

```
CWBDB_PCSTRING  
CWBDB_PCVARSTRING  
CWBDB_PCGRAPHIC  
CWBDB_PCVARGRAPHIC
```

cwbDB_SetCommitmentControl

Purpose: Set the commitment level for the database server to use when accessing data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetCommitmentControl(  
    cwbDB_ConnectionHandle connection,  
    unsigned short         commitmentLevel,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short commitmentLevel - input

Indicates the commitment level for server operations.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: Use one of the defined values:

CWBDB_NONE

CWBDB_CURSOR_STABILITY

CWBDB_CHANGE

CWBDB_ALL

The **cwbDB_ApplyAttributes** API must be called after **cwbDB_SetCommitmentControl** in order for the new commitment level to take affect.

cwbDB_SetConversionIndicator

Purpose: Sets the indicator that says whether data is to be converted between the client and host format.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetConversionIndicator(  
    cwbDB_FormatHandle format,  
    cwb_Boolean conversionIndicator,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_FormatHandle format - input

Handle to a data format object.

cwb_Boolean conversionIndicator - input

CWB_FALSE indicates no conversion. CWB_TRUE indicates conversion.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: The **cwbDB_ApplyAttributes** API must be called after **cwbDB_SetConversionIndicator** in order for the new value to take affect.

cwbDB_SetConvert65535

Purpose: Sets the indicator that says whether data marked with CCSID 65535 is to be converted between ASCII and EBCDIC. Data tagged with CCSID 65535 are binary data. Selecting to convert this data may cause conversion errors and possible data integrity problems. USE THIS API AT YOUR OWN RISK. Having stated that, it is important to note that some older data may have text data tagged with CCSID 65535. Also, some iSeries server tools still write data to files using CCSID 65535. Therefore, there may be appropriate times to use this API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetConvert65535(  
    cwbDB_ConnectionHandle connection,  
    cwb_Boolean convert65535indicator,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to a connection object.

cwb_Boolean convert65535indicator - input

CWB_FALSE indicates no conversion of binary data. CWB_TRUE indicates conversion of data will take place.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: None

cwbDB_SetCursorName

Purpose: Set the statement name to be used for this request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetCursorName(  
    cwbDB_RequestHandle request,  
    char *cursorName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *cursorName - input

Pointer to an ASCII string containing the cursor name being used for an SQL request.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_SetCursorReuse

Purpose: This API indicates to SQL what our future plans are for cursors when we close them. This is valid when there are multiple result sets.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetCursorReuse(  
    cwbDB_RequestHandle request,  
    unsigned short reuseIndicator,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short reuseIndicator - input

Input value for reuse indicator. This parameter should be one of the following values:

CWBDB_CLOSE_ALL_CURSORS - Close the cursor for all result sets.

CWBDB_CLOSE_CURRENT_CURSOR - Close the cursor for current result set only.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid reuseIndicator value.

Usage: When a cursor is opened against a stored procedure which has multiple result sets all result sets are opened and handled with the same cursor. When the cursor is opened it points to the first result set. When it is closed with the CWBDB_CLOSE_CURRENT_CURSOR option it closes the cursor and current result set. When it is opened again it points to the next result set until the last result set is closed.

When it is closed with the CWBDB_CLOSE_ALL_CURSORS option it closes the cursor and all result sets, so it cannot be opened again.

This API is not valid for NDB or catalog requests.

cwbDB_SetDateFormat

Purpose: Set the format for date data returned from the iSeries server. Date data on the iSeries server are stored encoded and are returned to the client as character strings. These character strings can be formatted in eight different ways:

Format name	Format	Example
Julian	yy/ddd	87/253
month day year	mm/dd/yy	10/12/87
day month year	dd/mm/yy	12/10/87
year month day	yy/mm/dd	87/10/12
USA	mm/dd/yyyy	10/12/1987
ISO	yyyy-mm-dd	1987-10-12
IBM Japan	yyyy-mm-dd	1987-10-12
IBM Europe	dd.mm.yyyy	12.10.1987

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetDateFormat(  
                                cwbDB_ConnectionHandle connection,  
                                unsigned short dateFormat,  
                                cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short dateFormat - input

Indicates the format of date data.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

CWBDB_INVALID_ARG_API

Value specified is not in range.

Usage: It is not valid to call this API after calling the **cwbDB_StartServer** API. Use one of the defined values:

Format name	Date format constant	Value
Julian	CWBDB_DATE_FMT_JUL	0
month day year	CWBDB_DATE_FMT_MDY	1
day month year	CWBDB_DATE_FMT_DMY	2
year month day	CWBDB_DATE_FMT_YMD	3
USA	CWBDB_DATE_FMT_USA	4
ISO	CWBDB_DATE_FMT_ISO	5
IBM Japan	CWBDB_DATE_FMT_JIS	6
IBM Europe	CWBDB_DATE_FMT_EUR	7

cwbDB_SetDateSeparator

Purpose: Set the character which separates the elements of date data returned from the iSeries server. Date data on the iSeries server are stored encoded and are returned to the client as character strings. These character strings can have one of five different date separator characters:

Date separator	Character	Example
Slash	/	03/17/94
Dash	-	03-17-94
Period	.	03.17.94
Comma	,	03,17,94
Blank		03 17 94

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetDateSeparator(  
    cwbDB_ConnectionHandle connection,  
    unsigned short dateSeparator,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short dateSeparator - input

Indicates the separator character for date fields.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

CWBDB_INVALID_ARG_API

Value specified is not in range.

Usage: It is not valid to call this API after calling the **cwbDB_StartServer** API. Use one of the defined values:

Date separator	Date separator constant
Slash	CWBDB_DATE_SEP_SLASH
Dash	CWBDB_DATE_SEP_DASH
Period	CWBDB_DATE_SEP_PERIOD
Comma	CWBDB_DATE_SEP_COMMA
Blank	CWBDB_DATE_SEP_BLANK

cwbDB_SetDecimalSeparator

Purpose: Set the character which separates the elements of decimal data returned from the iSeries server.

Decimal separator	Character	Example
Period	.	123.45
Comma	,	123,45

Syntax:

```
unsigned int CWB_ENTRY  cwbDB_SetDecimalSeparator(  
                        cwbDB_ConnectionHandle connection,  
                        unsigned short      decimalSeparator,  
                        cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short decimalSeparator - input

Indicates the desired decimal separator character.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

CWBDB_INVALID_ARG_API

Value specified is not in range.

Usage: It is not valid to call this API after calling the **cwbDB_StartServer** API. Use one of the defined values:

Time separator	Time separator constant
Period	CWBDB_DECIMAL_SEP_PERIOD
Comma	CWBDB_DECIMAL_SEP_COMMA

cwbDB_SetDescribeOption

Purpose: Set the describe option to determine what data is to be returned as a result of a describe.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetDescribeOption(  
    cwbDB_RequestHandle request,  
    unsigned short describeOption,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short describeOption - input

Long integer specifying the type of data to be returned on a describe operation.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid **describeOption** value.

Usage: Use one of the defined values:

CWBDB_DESC_ALIAS_NAMES

CWBDB_DESC_NAMES_ONLY

CWBDB_DESC_LABELS

This API is not valid for NDB or catalog requests.

cwbDB_SetDefaultSQLLibraryName

Purpose: Sets server attribute for the connection to indicate the default library name.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetDefaultSQLLibraryName(  
    cwbDB_ConnectionHandle connection,  
    char* libraryName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to a connection object..

char* libraryName, - input

Pointer to a character string up to 10 characters long that specifies the qualified library name to use on the SQL statement text when no library name is specified in the statement text. The default is 10 space characters

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

libraryName = NULL

CWBDB_STRING_ARG_TOO_LONG

libraryName > 10

Usage: This API may be called any time after the connection handle has been created, but if it is called after the server is started for that connection handle then the cwbDB_ApplyAttributes API must be called in order for the setting to take affect.

| **cwbDB_SetExtendedDataFormat**

| **Purpose:** This API indicates to SQL if it should build extended data format information.

| **Syntax:**

```
| unsigned int CWB_ENTRY cwbDB_SetExtendedDataFormat(  
|         cwbDB_RequestHandle    request,  
|         unsigned short         extendedFormatIndicator,  
|         cwbSV_ErrHandle        errorHandler);
```

| **Parameters:**

| **cwbDB_RequestHandle request - input**

| Handle to a request object.

| **unsigned short extendedFormatIndicator - input**

| Input value for extended format indicator. This parameter should be one of the following values:

- | • **CWBDB_USE_EXTENDED_FORMAT** — Indicates that extended data format will be used.
- | • **CWBDB_USE_NORMAL_FORMAT** — Indicates that base data format will be used.

| **cwbSV_ErrHandle errorHandler - input**

| Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

| **Return Codes:** The following list shows common return values.

| **CWB_OK**

| Successful completion.

| **CWB_INVALID_API_HANDLE**

| Invalid request handle.

| **CWBDB_INVALID_ARG_API**

| Invalid extendedFormat indicator value.

| **CWBDB_SERVER_FUNCTION_NOT_AVAILABLE**

| The host server is not at the required level to support this feature.

| **Usage:** This tells the host if it should build extended data format information. It can be included with any of the following flows or stored in the RPB:

- | • `cwbDB_ExecuteImmediate`
- | • `cwbDB_Prepate`
- | • `cwbDB_PrepateDescribe`
- | • `cwbDB_PrepateDescribeOpenFetch`

| Note that the host must know at prepare time to build the extended information. Also this call only tells the host to build the information. A call to **cwbDB_ReturnExtendedDataFormat** must be made before retrieving the information in order to actually get the extended info.

| The default value is to not build extended information.

| If the host server is not at the required level to support this feature, then this call will do nothing, the extended version of the data format will not be build, and a warning will be returned. Subsequent calls to get extended data will return default values.

cwbDB_SetFetchScrollOptions

Purpose: After using the **cwbDB_SetScrollableCursor**, this API is used to indicate how to scroll through the data.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFetchScrollOptions(  
    cwbDB_RequestHandle request,  
    unsigned short      scrollType,  
    unsigned long       relativeDistance,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short scrollType - input

Indicates type of scrolling to be performed.

unsigned long relativeDistance - input

If the **scrollType** indicates scrolling relative to the current cursor position, this parameter indicates the relative distance. For other **scrollType** values, this parameter is ignored.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid **scrollType** value.

Usage: Use one of the defined values:

```
CWBDB_SCROLL_DIRECT  
CWBDB_SCROLL_NEXT  
CWBDB_SCROLL_PREVIOUS  
CWBDB_SCROLL_FIRST  
CWBDB_SCROLL_LAST  
CWBDB_SCROLL_BEFORE_FIRST  
CWBDB_SCROLL_AFTER_LAST  
CWBDB_SCROLL_CURRENT  
CWBDB_SCROLL_RELATIVE
```

This API is not valid for NDB or catalog requests.

cwbDB_SetFieldName

Purpose: Set the field name to be used in a catalog request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFieldName(  
    cwbDB_RequestHandle request,  
    char *fieldName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *fieldName - input

Pointer to an ASCII string containing the field name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. This API is not valid for NDB or SQL requests.

cwbDB_SetFileAttributes

Purpose: Set the file attributes to be used as a qualifier for a list request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFileAttributes(  
    cwbDB_RequestHandle request,  
    unsigned short      fileAttributes,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short fileAttributes - input

Long integer that indicates attributes of files to be retrieved for a catalog request.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call. Use one of the defined values:

```
CWBDB_ALL_FILES_ATTRIBUTES  
CWBDB_PHYSICAL_FILES_ATTRIBUTES  
CWBDB_LOGICAL_FILES_ATTRIBUTES  
CWBDB_ODBC_TABLES_ATTRIBUTES  
CWBDB_ODBC_VIEWS_ATTRIBUTES
```

This API is not valid for NDB or SQL requests.

cwbDB_SetFileInfoOrdering

Purpose: Changes the ordering of the data returned by catalog requests.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFileInfoOrdering(  
                                     cwbDB_RequestHandle request,  
                                     unsigned short   fileInfoOrder,  
                                     cwbSV_ErrHandle  errorHandle
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long fileInfoOrdering - input

Long integer that indicates how the returned information is to be ordered.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call. Use one of the defined values:

```
CWBDB_DEFAULT_CATALOG_ORDERING  
CWBDB_ODBC_TABLE_ORDERING  
CWBDB_ODBC_TABLE_PRIVILEGE_ORDER
```

This API is not valid for NDB or SQL requests.

cwbDB_SetFileName

Purpose: Set the file name to be used as a qualifier for a list request. This is the short file name (system or server name).

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFileName(  
    cwbDB_RequestHandle request,  
    char *fileName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *fileName - input

Pointer to an ASCII string containing the file name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call. This API is not valid for SQL requests.

cwbDB_SetFileText

Purpose: Set the text description for a file that will be created through the API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFileText(  
    cwbDB_RequestHandle request,  
    char                *fileText,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *fileText - input

Pointer to an ASCII string that contains the text description to be used when creating a file.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests.

cwbDB_SetFileType

Purpose: Set the file type to be used as a qualifier for a list request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFileType(  
                                cwbDB_RequestHandle request,  
                                unsigned short      fileType,  
                                cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short fileType - input

Long integer that indicates type of files to be retrieved for a catalog request.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call. Use one of the defined values:

CWBDB_ALL_FILES

CWBDB_SOURCE_FILES

CWBDB_DATA_FILES

This API is not valid for NDB or SQL requests.

cwbDB_SetForeignKeyFileName

Purpose: Set the foreign key file name to be used in a request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetForeignKeyFileName(  
    cwbDB_RequestHandle request,  
    char *fileName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *fileName - input

Pointer to an ASCIIZ string containing the foreign key file name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. This API is not valid for NDB or SQL requests.

cwbDB_SetForeignKeyLibName

Purpose: Set the foreign key library name to be used in a request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetForeignKeyLibName(  
    cwbDB_RequestHandle request,  
    char *libName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *libName - input

Pointer to an ASCIIZ string containing the foreign key library name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. This API is not valid for NDB or SQL requests.

cwbDB_SetFormatName

Purpose: Set the record format name to be used in a request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetFormatName(  
    cwbDB_RequestHandle request,  
    char *formatName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *formatName - input

Pointer to an ASCII string containing the record format name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. This API is not valid for NDB or SQL requests.

cwbDB_SetHoldIndicator

Purpose: This API instructs SQL how to treat active statements (open cursors and prepared dynamic SQL statements) when a commit or rollback operation is performed. CWBDB_HOLD indicates that open cursors and prepared dynamic SQL statements will be preserved. CWBDB_WORK will cause open cursors to be closed and prepared dynamic SQL statement to be destroyed.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetHoldIndicator(  
    cwbDB_RequestHandle request,  
    unsigned short      holdIndicator,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short holdIndicator - input

Input value for hold indicator.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid **holdIndicator** value.

Usage: Use one of the defined values:

CWBDB_WORK

CWBDB_HOLD

This API is not valid for NDB or catalog requests.

cwbDB_SetIgnoreDecimalDataError

Purpose: Set an indicator that says whether to ignore or correct zoned decimal data errors.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetIgnoreDecimalDataError(  
    cwbDB_ConnectionHandle connection,  
    unsigned short         ignoreDecimalError,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short ignoreDecimalError - input

Indicates how decimal data errors will be treated.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: Use one of the defined values:

CWBDB_IGNORE_ERROR

CWBDB_CORRECT_ERROR

The **cwbDB_ApplyAttributes** API must be called after **cwbDB_SetIgnoreDecimalDataError** in order for the new value to take affect.

cwbDB_SetIndexType

Purpose: Set the type of index criteria to be used in a catalog request

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetIndexType(  
                                cwbDB_RequestHandle request,  
                                unsigned short      indexType,  
                                cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short indexType - input

Long integer that indicates index rule to be retrieved for a catalog request.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. Use one of the defined values:

```
CWBDB_UNIQUE_INDEX  
CWBDB_DUPLICATE_INDEX  
CWBDB_DUP_NULL_INDEX
```

This API is not valid for NDB or SQL requests.

cwbDB_SetLibraryName

Purpose: Set the library name to be used for the current database request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetLibraryName(  
    cwbDB_RequestHandle request,  
    char *libraryName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *libraryName - input

Pointer to an ASCIIZ string containing the library name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For SQL requests, this is the library that is used when locating an SQL package to be used for stored statements. For List and Native Database requests, this is the library containing objects on which to be operated.

cwbDB_SetLOBFieldThreshold

Purpose: Sets server attribute for the connection to indicate the threshold length for LOB fields.

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned long thresholdSize - input

Threshold where all FETCH result sets which contain a LOB field with a length which is less than or equal to the threshold length will be have the LOB data for the field returned in-line as part of the row data. If a LOB field in a result set has a length which is greater than the threshold, a LOB handle will be returned to the client on the FETCH request. Default is zero.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be called any time after the connection handle has been created and must be called before the server is started. This attribute cannot be changed after the server is started. The default value is zero.

cwbDB_SetLongFileName

Purpose: Set the long file name to be used as a qualifier for a list request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetLongFileName(  
    cwbDB_RequestHandle request,  
    char *longFileName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *longFileName - input

Pointer to an ASCII string containing the long file name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call. This API is not valid for NDB or SQL requests.

cwbDB_SetMaximumMembers

Purpose: Set the maximum number of members for creating a file through the API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetMaximumMembers(  
    cwbDB_RequestHandle request,  
    signed short        maxMembers,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

signed short maxMembers - input

Input value for maximum number of members. A value of -1 for this parameter indicates no maximum.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests.

cwbDB_SetMemberName

Purpose: Set the member name to be used in a request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetMemberName(  
    cwbDB_RequestHandle request,  
    char *memberName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *memberName - input

Pointer to an ASCII string containing the member name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. This API is also used for NDB requests when operating on a database file member. This API is not valid for SQL requests.

cwbDB_SetMemberText

Purpose: Set the text description for a member that will be added through the API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetMemberText(  
    cwbDB_RequestHandle request,  
    char *memberText,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *memberText - input

Pointer to an ASCII string that contains the text description to be used when adding a member.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests.

cwbDB_SetNamingConvention

Purpose: Set the naming convention (SQL or iSeries server) to be used by the database access server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetNamingConvention(  
    cwbDB_ConnectionHandle connection,  
    unsigned short         newNamingConvention,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server

unsigned short newNamingConvention - input

Indicates the type of naming convention to use. SQL naming convention (library.table) or iSeries native naming convention (library/table).

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

CWBDB_INVALID_ARG_API

Invalid naming convention value.

Usage: Use one of the defined values:

CWBDB_PERIOD_NAME_CONV

CWBDB_SLASH_NAME_CONV

The **cwbDB_ApplyAttributes** API must be called after **cwbDB_SetNamingConvention** in order for the new naming convention to take affect.

cwbDB_SetNLSS

Purpose: Sets the National Language Sort Sequence (NLSS) attribute of the Data Access server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetNLSS(  
    cwbDB_ConnectionHandle connection,  
    unsigned short          NLSSTypeID,  
    char                   *tableOrLangID,  
    char                   *library,  
    cwbSV_ErrHandle       errorHandle );
```

Parameters:

cwbDB_ConnectionHandle - input

Connection through which the attribute is to be set

unsigned short NLSSTypeID - input

The type of NLSS attribute. Possible values:

CWBDB_NLSS_SORT_HEX

CWBDB_NLSS_SORT_SHARED

CWBDB_NLSS_SORT_UNIQUE

CWBDB_NLSS_SORT_USER

char *tableOrLangID - input

Depends on value of the **NLSSType** parameter (above).

CWBDB_NLSS_SORT_HEX

This parameter is not used

CWBDB_NLSS_SORT_SHARED or CWBDB_NLSS_SORT_UNIQUE

This parameter represents the language feature code attribute ID for the server. It is a required parameter.

CWBDB_NLSS_SORT_USER

This parameter represents the NLSS table name attribute. It is a required parameter.

char *library - input

Depends on value of the **NLSSType** parameter (above).

CWBDB_NLSS_SORT_HEX

This parameter is not used.

CWBDB_NLSS_SORT_SHARED or CWBDB_NLSS_SORT_UNIQUE

This parameter is not used

CWBDB_NLSS_SORT_USER

This parameter represents the NLSS library name attribute. It is an optional parameter.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid connection handle.

CWBDB_INVALID_ARG_API

Invalid type, language ID, or table.

Usage: The **cwbDB_ApplyAttributes** API must be called after **cwbDB_SetNLSS** in order for the new sort sequence to take affect.

cwbDB_SetNullable

Purpose: Set the nullable indicator for a special column.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetNullable(  
    cwbDB_RequestHandle request,  
    unsigned short nullableInd,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short nullableInd - input

Integer that indicates whether special column is null capable.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call. Use one of the defined values:

CWBDB_NOT_NULLABLE

CWBDB_NULLABLE

This API is not valid for NDB or SQL requests.

cwbDB_SetOverrideInformation

Purpose: Set the overriding library, file, and member for an override database operation.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetOverrideInformation(  
    cwbDB_RequestHandle request,  
    char                *overrideLibraryName,  
    char                *overrideFileName,  
    char                *overrideMemberName,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *overrideLibraryName - input

Pointer to an ASCIIZ string that contains the overriding library name.

char *baseFileName - input

Pointer to an ASCIIZ string that contains the overriding file name.

char *overrideMemberName - input

Pointer to an ASCIIZ string that contains the overriding member name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is used in preparation for **cwbDB_OverrideFile**. This API is not valid for List or SQL requests.

cwbDB_SetPackageName

Purpose: Set the SQL package name for a database request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetPackageName(  
    cwbDB_RequestHandle request,  
    char *packageName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *packageName - input

Pointer to an ASCIIZ string containing the SQL package name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: For catalog requests, this API is used prior to **cwbDB_RetrievePackageInformation** or **cwbDB_RetrievePackageStatementInformation**. For SQL requests, this API is used to set the name of the SQL package to be used for preparing or executing SQL statements. This is optional for SQL requests. This API is not valid for NDB requests.

cwbDB_SetParameterMarkerBlock

Purpose: Provides the data to be used for the parameter markers contained in a prepared statement for a block of rows.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetParameterMarkerBlock(  
    cwbDB_RequestHandle request,  
    unsigned long      numberOfRows,  
    cwbDB_FormatHandle format,  
    void               *dataPointer,  
    signed short       *indicators,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long numberOfRows - input

Number of sets of parameter marker data that is in the dataBuffer.

cwbDB_FormatHandle format - input

Handle to the format of the data being provided.

void *dataBuffer - input

Pointer to a buffer containing the data to be used for the parameter markers.

signed short *indicators - input

Pointer to a buffer containing the null indicators. If the value of the indicator is less than zero, the value for the corresponding parameter marker is null.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_SetParameterMarkers

Purpose: Provides the data to be used for the parameter markers contained in a prepared statement.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetParameterMarkers(  
    cwbDB_RequestHandle request,  
    cwbDB_FormatHandle  format,  
    void                *dataBuffer,  
    signed short        *indicators,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbDB_FormatHandle *format - input

Handle to the format of the data being provided.

void *dataBuffer - input

Pointer to a buffer containing the data to be used for the parameter markers.

signed short *indicators - input

Pointer to a buffer containing the null indicators. If the value of the indicator is less than zero, the value for the corresponding parameter marker is null.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_SetPrepareOption

Purpose: Set the option for doing a normal or enhanced prepare. Doing an enhanced prepare will search the specified SQL package for the given statement. If it is found, the statement will be used. If not, the statement will be prepared.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetPrepareOption(  
    cwbDB_RequestHandle request,  
    unsigned short      prepareOption,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short prepareOption - input

Long integer specifying the type of prepare to be performed.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid **prepareOption** value.

Usage: Use one of the defined values:

CWBDB_NORMAL_PREPARE

CWBDB_ENHANCED_PREPARE

This API is not valid for NDB or catalog requests.

cwbDB_SetPrimaryKeyFileName

Purpose: Set the primary key file name to be used in a request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetPrimaryKeyFileName(  
    cwbDB_RequestHandle request,  
    char *fileName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *fileName - input

Pointer to an ASCIIZ string containing the primary key file name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. This API is not valid for NDB or SQL requests.

cwbDB_SetPrimaryKeyLibName

Purpose: Set the primary key library name to be used in a request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetPrimaryKeyLibName(  
    cwbDB_RequestHandle request,  
    char *libName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *libName - input

Pointer to an ASCIIZ string containing the primary key library name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to a **cwbDB_Retrieve*** API call for a catalog request. This API is not valid for NDB or SQL requests.

cwbDB_SetQueryTimeoutValue

Purpose: Sets the query timeout value contained in the RPB.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetQueryTimeoutValue(  
    cwbDB_RequestHandle request,  
    long timeout,  
    cwbSV_ErrHandle errorHandler);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object. This api is only valid for an SQL request.

long timeout - input

The timeout value greater than zero. The special value -1 indicates a value of *NOMAX.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_PARAMETER_ERROR

Timeout not greater than zero or -1.

Usage: The cwbDB_StoreRequestParameters API must be called in order for the setting to take affect.

cwbDB_SetRDBName

Purpose: Set the Relational Database (RDB) name for a catalog request. This is the RDB for which information is being requested.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetRDBName(  
    cwbDB_RequestHandle request,  
    char *RDBName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *RDBName - input

Pointer to an ASCII string containing the RDB name.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is used prior to **cwbDB_RetrieveDBInformation**. This API is not valid for SQL or NDB requests.

cwbDB_SetRecordLength

Purpose: Set the record length in preparation for creating a file through the API.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetRecordLength(  
    cwbDB_RequestHandle request,  
    unsigned long      recordLength,  
    cwbSV_ErrHandle   errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned long recordLength - input

Length of records to be contained in the file to be created.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for List or SQL requests.

cwbDB_SetScrollableCursor

Purpose: Indicate whether the cursor used by this request is scrollable.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetScrollableCursor(  
    cwbDB_RequestHandle request,  
    unsigned short      scrollIndicator,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short scrollIndicator - input

Input value for scroll indicator.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid scrollIndicator value.

Usage: Use one of the defined values:

CWBDB_CURSOR_STATIC_SCROLLABLE

CWBDB_CURSOR_NOT_SCROLLABLE

CWBDB_CURSOR_SCROLLABLE

This API is not valid for NDB or catalog requests.

cwbDB_SetStatementName

Purpose: Set the statement name to be used for this request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetStatementName(  
    cwbDB_RequestHandle request,  
    char *statementName,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *statementName - input

Pointer to an ASCII string containing the statement name being used for an SQL request.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_SetStatementText

Purpose: Set the statement text to be used for this request.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetStatementText(  
    cwbDB_RequestHandle request,  
    char *statementText,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

char *statementText - input

Pointer to an ASCIIZ string containing the statement text being used for an SQL request.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_SetStatementType

Purpose: Set the type of SQL statement for which information is being requested.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetStatementType(  
    cwbDB_RequestHandle request,  
    unsigned short      statementType,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short statementType - input

Long integer that indicates type of SQL statement being used for a catalog request.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Usage: This API may be used prior to making a **cwbDB_RetrieveSQLPackageStatement** API call. Use one of the defined values:

```
CWBDB_ALL_STATEMENTS  
CWBDB_DECLARE_STATEMENTS  
CWBDB_SELECT_STATEMENTS  
CWBDB_EXEC_STATEMENTS
```

This API is not valid for NDB or SQL requests.

cwbDB_SetStaticCursorResultSetThreshold

Purpose: Sets threshold for static cursor result set size.

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object. This api is only valid for an SQL request.

unsigned long thresholdSize - input

Threshold that limits the number of records in a temporary record set of a static cursor. Valid range is 1 - 2147483647 (2GB- 1)). Default value is 2147483647.

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the cwbSV_CreateErrHandle API. The messages may be retrieved through the cwbSV_GetErrText API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_INVALID_ARG_API

Invalid reuseIndicator value.

Usage: This API is not valid for NDB or catalog requests.

cwbDB_SetStreamFetchSyncCount

Purpose: Set the number of 32Kb blocks sent from the server to the client during a stream fetch before a synchronizing handshake is required.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetStreamFetchSyncCount(  
    cwbDB_RequestHandle request,  
    unsigned short      syncCount,  
    cwbSV_ErrHandle     errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

unsigned short syncCount - input

Unsigned short integer that indicates how many 32Kb flows from the server will happen before a synchronizing handshake will happen.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

CWBDB_STREAM_FETCH_NOT_COMPLETE

Stream fetch in process.

Usage: This API is not valid for NDB or Catalog requests. This API must be called before the **cwbDB_DynamicStreamFetch** or the **cwbDB_ExtendedDynamicStreamFetch** API is called.

cwbDB_SetTimeFormat

Purpose: Set the format for time data returned from the iSeries server. Time data on the iSeries server are stored encoded and are returned to the client as character strings. These character strings can be formatted in five different ways:

Format name	Format	Example
Hours minutes seconds	hh:mm:ss	13:30:05
USA	hh:mm AM or PM	1:30 PM
ISO	hh.mm.ss	13:30:05
IBM Europe	hh.mm.ss	13:30:05
IBM Japan	hh:mm:ss	13:30:05

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetTimeFormat(  
    cwbDB_ConnectionHandle connection,  
    unsigned short timeFormat,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short timeFormat - input

Indicates the format of time data.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

CWBDB_INVALID_ARG_API

Value specified is not in range.

Usage: It is not valid to call this API after calling the **cwbDB_StartServer** API. Use one of the defined values:

Format name	Time format constant
Hours minutes seconds	CWBDB_TIME_FMT_HMS
USA	CWBDB_TIME_FMT_USA
ISO	CWBDB_TIME_FMT_ISO
IBM Europe	CWBDB_TIME_FMT_EUR
IBM Japan	CWBDB_TIME_FMT_JIS

cwbDB_SetTimeSeparator

Purpose: Set the character which separates the elements of time data returned from the iSeries server. Time data on the iSeries server are stored encoded and are returned to the client as character strings. These character strings can have one of four different time separator characters:

Date separator	Character	Example
Colon	:	11:10:03
Period	.	11.10.03
Comma	,	11,10,03
Blank		11 10 03

Syntax:

```
unsigned int CWB_ENTRY cwbDB_SetTimeSeparator(  
    cwbDB_ConnectionHandle connection,  
    unsigned short timeSeparator,  
    cwbSV_ErrHandle errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned short timeSeparator - input

Indicates the time data separator character.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

CWBDB_INVALID_ARG_API

Value specified is not in range.

Usage: It is not valid to call this API after calling the **cwbDB_StartServer** API. Use one of the defined values:

Time separator	Time separator constant
Colon	CWBDB_TIME_SEP_COLON
Period	CWBDB_TIME_SEP_PERIOD
Comma	CWBDB_TIME_SEP_COMMA
Blank	CWBDB_TIME_SEP_BLANK

cwbDB_StartServer

Purpose: Starts the communication between the client and the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_StartServer(  
    cwbDB_ConnectionHandle connection,  
    cwbSV_ErrHandle        errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: None

cwbDB_StartServerDetailed

Purpose: Starts the communication between the client and the iSeries server. Returns a more detailed return code than **cwbDB_StartServer**, but otherwise the same.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_StartServerDetailed(  
    cwbDB_ConnectionHandle connection,  
    unsigned long          *returnCode,  
    cwbSV_ErrHandle       errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

unsigned long *returnCode - output

Pointer to an unsigned long to receive the detailed return code.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: None.

cwbDB_StopServer

Purpose: Ends the communication between the client and the iSeries server.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_StopServer(  
    cwbDB_ConnectionHandle connection,  
    cwbSV_ErrHandle        errorHandle);
```

Parameters:

cwbDB_ConnectionHandle connection - input

Handle to connection to iSeries database access server.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: None

cwbDB_StoreRequestParameters

Purpose: Sends the current parameters to the iSeries server to be stored by the database access server. Those parameters can then be used by the request on subsequent function calls.

Syntax:

```
unsigned int CWB_ENTRY cwbDB_StoreRequestParameters(  
    cwbDB_RequestHandle request,  
    cwbSV_ErrHandle    errorHandle);
```

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

cwbSV_ErrHandle errorHandle - input

Any returned messages will be written to this object. It is created with the **cwbSV_CreateErrHandle** API. The messages may be retrieved through the **cwbSV_GetErrText** API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Incorrect connection handle.

Usage: This API is used to store a set of parameters in a buffer on the iSeries server. This is useful if there is a set of common parameters that are to be used for multiple functions. The API allows the application to reduce the amount of data that needs to flow in order to perform all of the requests.

cwbDB_WriteLOBData

Purpose: Write LOB Data.

Parameters:

cwbDB_RequestHandle request - input

Handle to a request object.

void* dataPointer

unsigned long locator - input

unsigned short ccsid - input

unsigned long size - input

unsigned long start - input

cwbSV_ErrHandle errorHandler - input

Any returned messages will be written to this object. It is created with the `cwbSV_CreateErrHandle` API. The messages may be retrieved through the `cwbSV_GetErrText` API. If the parameter is set to zero, no messages will be retrievable.

Return Codes: The following list shows common return values.

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid request handle.

Example: Using SQL to access database functions

```
////////////////////////////////////
//
// PRFTST.CPP
// CLIENT ACCESS DATA ACCESS SAMPLE PROGRAM - Block Fetch a whole table
// Usage: prftst systemname blocksize limit
//      systemname - name of the iSeries to run against
//      blocksize - number of rows to bring down in each fetch call
//                  default: 1 row
//      limit - total number of rows to bring down
//              default: INT_MAX
// Input file: prftst.qry: Put the text of your input query in
//               an ASCII file of this name. Limit: 500 characters,
//               unless you change it. (See MAXSIZE constant.)
//               Example: SELECT * FROM QIWS.QCUSTCDT
// Usage notes: If the blocksize exceeds the number of rows in the
//               table, the entire table is fetched.
//
//               Link with CWBAPI.LIB
//
////////////////////////////////////

#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <limits.h>

#include "CWDBD.H" // Header for Database access API's
#include "CWBSV.H" // Header for Serviceability API's

void scene18( char*, int, int );

void main( int argc, char *argv[] )
{
char sys[15] = "SYSTEMXX";
int block = 1;
int limit = INT_MAX;

    if ( argc > 1 )
    {
        for( unsigned int i = 0; i<=strlen(argv[1]); i++ )
            sys[i] = (char) toupper(argv[1][i]);
    }

    if ( argc > 2 )
    {
        block = atoi(argv[2]);
    }

    if ( argc > 3 )
    {
        limit = atoi(argv[3]);
    }
    scene18(sys, block, limit);
    return;
}

void scene18( char *systemName, int blockSize, int fetchLimit )
{
    FILE *infile, *outfile;
    outfile = fopen("prftst.out","w");
    char *cursorName = "CURSOR1";
    char *statementName = "BTDB018";
    const int MAXSIZE = 500;
    char statementText[MAXSIZE] = "";
    unsigned int rc;
    int rowCount = 0;
    unsigned long dataLength = 0;
    char ch;
```

```

cwbDB_FormatHandle myFmt;
cwbDB_ConnectionHandle Conn;
cwbSV_ErrHandle errorHandler;
cwbDB_RequestHandle SQLReq;

cwbDB_DataHandle myData, ind, msgid, first, sec;
unsigned short hClass;
signed long hCode;

// Read the input file

int count = 0;
if ( (infile = fopen("prftst.qry","r")) != NULL) {
    while ( (ch = getc(infile)) != EOF && ch != '\n' && count < MAXSIZE ) {
        statementText[count] = ch;
        count++;
    }
    count++;
    statementText[count] = '\n';
} else {
    cout << "Need input query parameter in prftst.qry." << endl;
    return;
}

cout << "Block Fetch with data conversion" << endl << endl;

// Create a necessary handles

cwbDB_CreateDataHandle(&myData,; errorHandler);
cwbDB_CreateDataHandle(&ind,; errorHandler);
cwbDB_CreateDataHandle(&msgid,; errorHandler);
cwbDB_CreateDataHandle(&first,; errorHandler);
cwbDB_CreateDataHandle(&sec,; errorHandler);
cwbSV_CreateErrHandle(&errorHandler);
cwbDB_CreateConnectionHandle(systemName, &Conn,; errorHandler);
cwbDB_CreateSQLRequestHandle(Conn, &SQLReq,; errorHandler);
cwbDB_CreateDataFormatHandle(Conn, &myFmt,; errorHandler);

cout << "Starting data access server on system: " << systemName << endl;

// Start the database access server

if ((rc = cwbDB_StartServer(Conn, errorHandler)) != 0)
{
    cout << "Bad return code from the startServer call: " << rc << endl;
    return;
}

// ***** Setup - prepare statement *****

if ((rc = cwbDB_SetStatementName(SQLReq, statementName, errorHandler)) != 0)
{
    cout << "FAIL - set statement name failed with return code: " << rc
        << endl << endl;
    return;
}

if ((rc = cwbDB_SetCursorName(SQLReq, cursorName, errorHandler)) != 0)
{
    cout << "FAIL - set cursor name failed with return code: "
        << rc << endl << endl;
    return;
}

if ((rc = cwbDB_StoreRequestParameters(SQLReq, errorHandler)) != 0)
{
    cout << "FAIL - store parameters failed with return code: " << rc
        << endl << endl;
}

```

```

    return;
}

if ((rc = cwbdb_SetStatementText(SQLReq, statementText, errorHandle)) != 0)
{
    cout << "FAIL - set statement text failed with return code: " << rc
        << endl << endl;
    return;
}

if ((rc = cwbdb_Prepare(SQLReq, errorHandle)) != 0)
{
    cout << "FAIL - prepare request failed: " << rc
        << endl << endl;
    return;
}

// ***** Open cursor *****

if ((rc = cwbdb_Open(SQLReq, CWBDB_READ, errorHandle)) != 0)
{
    cout << "FAIL - open request failed: " << rc
        << endl << endl;
    return;
}

// ***** Fetch data *****

if ((rc = cwbdb_SetCursorName(SQLReq, cursorName, errorHandle)) != 0)
{
    cout << "FAIL - set cursor name failed with return code: "
        << rc << endl << endl;
    return;
}

cwbdb_SetConversionIndicator(myFmt, 1, errorHandle);

// Loop through the block fetch until the limit is reached.
// If the limit is bigger than the total number of rows in the table,
// the fetch will eventually fail.

while (rowCount < fetchLimit) {

    if ((cwbdb_ReturnData(SQLReq, myData, ind, myFmt, errorHandle)) != 0)
    {
        cout << "FAIL - request for data to be returned failed: " << rc
            << endl << endl;
        return;
    }

    if ((rc = cwbdb_ReturnHostErrorInfo(SQLReq, &hClass, &hCode, msgid, first, sec,
        errorHandle)) != 0)
    {
        cout << "FAIL - request for return host error info failed: " << rc
            << endl << endl;
    }

    if ((rc = cwbdb_SetBlockCount(SQLReq, blockSize, errorHandle)) != 0)
    {
        cout << "FAIL - set block size failed with return code: " << rc
            << endl << endl;
        return;
    }

    cout << "Fetching a block of " << dec << blockSize << "." << endl;

    if ((rc = cwbdb_Fetch(SQLReq, errorHandle)) != 0)
    {
        char* firsttxt;
        char* sectxt;
        char** pfirsttxt = &firsttxt;
        char** psectxt = &sectxt;
        cwbdb_GetDataPointer(first, pfirsttxt, errorHandle);
        cwbdb_GetDataPointer(sec, psectxt, errorHandle);
    }
}

```

```

cout << endl << "Host message class: " << hClass << endl;
cout << endl << "Host message code: " << hCode << endl;
cout << "FIRST LEVEL TEXT: " << endl;
cout << firsttxt << endl << endl;
cout << "SECOND LEVEL TEXT: " << endl;
cout << sectxt << endl << endl;

    break;
}
else
{
    cout << "Fetch call ENDED." << endl;

    rowCount+=blockSize;

    cout << "Total rows fetched so far: " << dec << rowCount << "." << endl << endl;

    if (blockSize <= 10) {
        char *theData = NULL;
        char **pmyData = &theData;
        unsigned long len;
        cwbdb_GetDataPointer(myData, pmyData, errorHandle);
        cwbdb_GetDataLength(myData, &len,; errorHandle);
        cout << "Fetched data: " << endl;
        cout.write( theData, len );
        cout << endl;
    }
}

} // end while

// Stop the database access server
cwbdb_StopServer(Conn, errorHandle);

// Delete all the handles
cwbdb_DeleteDataHandle(myData, errorHandle);
cwbdb_DeleteDataHandle(ind, errorHandle);
cwbdb_DeleteDataHandle(msgid, errorHandle);
cwbdb_DeleteDataHandle(first, errorHandle);
cwbdb_DeleteDataHandle(sec, errorHandle);
cwbdb_DeleteDataFormatHandle(myFmt, errorHandle);
cwbdb_DeleteConnectionHandle(Conn, errorHandle);
cwbdb_DeleteSQLRequestHandle(SQLReq, errorHandle);
cwbsv_DeleteErrHandle(errorHandle);
}

```

Chapter 6. Java programming

The **Java** programming language, which was defined by Sun, enables the development of portable Web-based applications.

See the IBM Toolbox for Java

The IBM Toolbox for Java, which is shipped with iSeries Access for Windows, provides Java classes for accessing iSeries resources. IBM Toolbox for Java uses the iSeries Access for Windows Host Servers as access points to the system. However, you do not need iSeries Access for Windows to use IBM Toolbox for Java. Use the Toolbox to write applications that run independent of iSeries Access for Windows.

Note: IBM Toolbox for Java interface behaviors such as security and tracing may differ from those of other iSeries Access for Windows interfaces.

Chapter 7. ActiveX programming

ActiveX automation is a programming technology that is defined by Microsoft.

iSeries Access for Windows provides the following methods for accessing iSeries resources by using ActiveX automation:

Automation objects:

These objects provide support for:

- Accessing iSeries data queues
- Calling iSeries system application programming interfaces and user programs
- Managing iSeries connections and validating security
- Running CL commands on the iSeries server
- Performing data-type and code-page conversions
- Performing database transfers
- Interfacing with host emulation sessions

“iSeries Access for Windows OLE DB Provider” on page 555:

Call the iSeries Access for Windows OLE DB Provider, by using Microsoft’s ActiveX Data Objects (ADO), to access the following iSeries server resources:

- The iSeries database, through record-level access
- The iSeries database, through SQL
- SQL stored procedures
- Data queues
- Programs
- CL commands

Custom controls:

ActiveX custom controls are provided for:

- iSeries data queues
- iSeries CL commands
- iSeries system names for previously connected systems
- iSeries Navigator

Programmer’s Toolkit:

For detailed information on ActiveX support for iSeries Access for Windows, see the **ActiveX** topic in the **Programmer’s Toolkit** component of iSeries Access for Windows. It includes complete documentation of ADO and ActiveX automation objects, and links to ActiveX information resources.

How to access the ActiveX topic:

1. Ensure that the **Programmer’s Toolkit** is installed (see “Installing the Programmer’s Toolkit” on page 12).
2. Launch the **Programmer’s Toolkit** (see “Launching the Programmer’s Toolkit” on page 12).
3. Select the **Overview** topic.
4. Select **Programming Technologies**.
5. Select **ActiveX**.



Printed in U.S.A.