



IBM i

IBM i Access for Windows: Programming

7.1







IBM i

IBM i Access for Windows: Programming

*7.1*

**Note**

Before using this information and the product it supports, read the information in “Notices,” on page 579.

| This edition applies to IBM i 7.1 of IBM i Access for Windows (product number 5770-XE1) and to all subsequent  
| releases and modifications until otherwise indicated in new editions. This version does not run on all reduced  
| instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1999, 2010.

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

---

# Contents

## IBM i Access for Windows :

### Programming . . . . . 1

What's new for IBM i 7.1 . . . . . 1

PDF file for IBM i Access for Windows: Programming 2

IBM i Access for Windows C/C++ APIs . . . . . 2

IBM i Access for Windows C/C++ APIs overview 2

API groups, header files, import libraries, and DLLs . . . . . 2

Programmer's Toolkit . . . . . 4

Install the Programmer's Toolkit . . . . . 5

Launch the Programmer's Toolkit . . . . . 5

IBM i name formats for connection APIs . . . . . 5

OEM, ANSI, and Unicode considerations . . . . . 6

Use a single IBM i Access for Windows API type . . . . . 8

Use mixed IBM i Access for Windows API types . . . . . 9

Write a generic IBM i Access for Windows application . . . . . 9

Obsolete IBM i Access for Windows APIs. . . . . 9

Obsolete Communications APIs . . . . . 10

Obsolete Data Queues APIs . . . . . 10

Obsolete Remote Command/Distributed

Program Call APIs . . . . . 11

Obsolete Security APIs. . . . . 11

Obsolete Serviceability APIs . . . . . 12

Obsolete System Object Access (SOA) API

Obsolete National Language Support (NLS)

APIs. . . . . 12

Return codes and error messages . . . . . 13

IBM i Access for Windows return codes

that correspond to operating system errors . 13

IBM i Access for Windows return codes . . 14

IBM i Access for Windows

component-specific return codes . . . . . 21

IBM i Access for Windows Administration APIs 32

Administration APIs list . . . . . 33

cwbAD\_GetClientVersion. . . . . 33

cwbAD\_GetProductFixLevel. . . . . 33

cwbAD\_IsComponentInstalled . . . . . 34

cwbAD\_IsOpNavPluginInstalled . . . . . 37

Example: Administration APIs . . . . . 38

IBM i Access for Windows Communications and

Security APIs. . . . . 43

System object attributes . . . . . 44

System object attributes list . . . . . 44

Communications and security: Create and

delete APIs . . . . . 47

cwbCO\_CreateSystem . . . . . 47

cwbCO\_CreateSystemLike . . . . . 48

cwbCO\_DeleteSystem . . . . . 50

Communications and security: Connect and

disconnect APIs . . . . . 50

cwbCO\_Connect. . . . . 50

cwbCO\_Disconnect. . . . . 52

cwbCO\_GetConnectTimeout. . . . . 53

cwbCO\_GetPersistenceMode. . . . . 54

cwbCO\_IsConnected . . . . . 55

cwbCO\_SetConnectTimeout . . . . . 56

cwbCO\_SetPersistenceMode . . . . . 57

cwbCO\_Verify . . . . . 58

Communication and security: Security

validation and data APIs . . . . . 59

cwbCO\_ChangePassword. . . . . 59

cwbCO\_GetDefaultUserMode . . . . . 61

cwbCO\_GetFailedSignons . . . . . 61

cwbCO\_GetPasswordExpireDate . . . . . 62

cwbCO\_GetPrevSignonDate . . . . . 63

cwbCO\_GetPromptMode . . . . . 65

cwbCO\_GetSignonDate . . . . . 65

cwbCO\_GetUserIDEx . . . . . 66

cwbCO\_GetValidateMode . . . . . 67

cwbCO\_GetWindowHandle . . . . . 68

cwbCO\_HasSignedOn . . . . . 68

cwbCO\_SetDefaultUserMode . . . . . 69

cwbCO\_SetPassword . . . . . 71

cwbCO\_SetPromptMode . . . . . 72

cwbCO\_SetUserIDEx . . . . . 73

cwbCO\_SetWindowHandle . . . . . 74

cwbCO\_SetValidateMode . . . . . 75

cwbCO\_Signon . . . . . 76

cwbCO\_VerifyUserIDPassword . . . . . 77

Communications and security: Get and set

attribute APIs. . . . . 79

cwbCO\_CanModifyDefaultUserMode. . . . . 79

cwbCO\_CanModifyIPAddress . . . . . 80

cwbCO\_CanModifyIPAddressLookupMode 80

cwbCO\_CanModifyPersistenceMode . . . . . 81

cwbCO\_CanModifyPortLookupMode. . . . . 82

cwbCO\_CanModifyUseSecureSockets. . . . . 83

cwbCO\_GetDescription . . . . . 83

cwbCO\_GetHostCCSID . . . . . 84

cwbCO\_GetHostVersionEx . . . . . 85

cwbCO\_GetIPAddress . . . . . 86

cwbCO\_GetIPAddressLookupMode . . . . . 87

cwbCO\_GetPortLookupMode . . . . . 88

cwbCO\_GetSystemName . . . . . 88

cwbCO\_IsSecureSockets . . . . . 89

cwbCO\_SetIPAddress . . . . . 90

cwbCO\_SetIPAddressLookupMode . . . . . 91

cwbCO\_SetPortLookupMode . . . . . 93

cwbCO\_UseSecureSockets . . . . . 94

Defines for cwbCO\_Service . . . . . 95

Differences between cwbCO\_Signon and

cwbCO\_VerifyUserIDPassword . . . . . 96

Similarities between cwbCO\_Signon and

cwbCO\_VerifyUserIDPassword . . . . . 96

Communications: Create and delete APIs . . 96

cwbCO\_CreateSysListHandle . . . . . 96

cwbCO\_CreateSysListHandleEnv . . . . . 97

cwbCO\_DeleteSysListHandle . . . . . 98

cwbCO_GetNextSysName . . . . .	98	cwbDQ_SetSenderID . . . . .	150
cwbCO_GetSysListSize . . . . .	100	Data Queues: Read and write APIs . . . . .	151
Communications: System information APIs	100	cwbDQ_CreateData . . . . .	151
cwbCO_GetActiveConversations . . . . .	100	cwbDQ_DeleteData . . . . .	151
cwbCO_GetConnectedSysName . . . . .	101	cwbDQ_GetConvert . . . . .	152
cwbCO_GetDefaultSysName . . . . .	102	cwbDQ_GetData . . . . .	152
cwbCO_IsSystemConfigured . . . . .	103	cwbDQ_GetDataAddr . . . . .	153
cwbCO_IsSystemConfiguredEnv . . . . .	103	cwbDQ_GetDataLen . . . . .	154
cwbCO_IsSystemConnected . . . . .	104	cwbDQ_GetKey . . . . .	155
Communications: Configured environments		cwbDQ_GetKeyLen . . . . .	155
information . . . . .	105	cwbDQ_GetRetDataLen . . . . .	156
cwbCO_GetActiveEnvironment . . . . .	105	cwbDQ_GetRetKey . . . . .	157
cwbCO_GetEnvironmentName . . . . .	106	cwbDQ_GetRetKeyLen . . . . .	157
cwbCO_GetNumberOfEnvironments . . . . .	107	cwbDQ_GetSearchOrder . . . . .	158
Communications: Environment and		cwbDQ_GetSenderInfo . . . . .	159
connection information . . . . .	107	cwbDQ_SetConvert . . . . .	160
cwbCO_CanConnectNewSystem . . . . .	107	cwbDQ_SetData . . . . .	160
cwbCO_CanModifyEnvironmentList . . . . .	108	cwbDQ_SetDataAddr . . . . .	161
cwbCO_CanModifySystemList . . . . .	108	cwbDQ_SetKey . . . . .	162
cwbCO_CanModifySystemListEnv . . . . .	109	cwbDQ_SetSearchOrder . . . . .	163
cwbCO_CanSetActiveEnvironment . . . . .	109	Example: Using Data Queues APIs . . . . .	163
Example: Using IBM i Access for Windows		IBM i Access for Windows data transformation	
communications APIs . . . . .	110	and National Language Support (NLS) APIs . . . . .	165
IBM i Data Queues APIs . . . . .	121	IBM i Access for Windows data	
Data queues . . . . .	122	transformation APIs . . . . .	165
Ordering data queue messages . . . . .	122	IBM i Access for Windows data	
Work with data queues . . . . .	122	transformation API list . . . . .	165
Typical use of data queues . . . . .	123	Example: Using data transformation APIs . . . . .	182
Data Queues: Create, delete, and open APIs	124	IBM i Access for Windows National	
cwbDQ_CreateEx . . . . .	124	Language Support (NLS) APIs . . . . .	183
cwbDQ_DeleteEx . . . . .	126	Coded character sets . . . . .	184
cwbDQ_OpenEx . . . . .	127	IBM i Access for Windows general NLS	
Data Queues: Accessing data queues APIs	129	APIs list . . . . .	184
cwbDQ_AsyncRead . . . . .	129	IBM i Access for Windows conversion	
cwbDQ_Cancel . . . . .	130	NLS APIs list . . . . .	191
cwbDQ_CheckData . . . . .	131	IBM i Access for Windows dialog-box	
cwbDQ_Clear . . . . .	132	NLS API list . . . . .	203
cwbDQ_Close . . . . .	133	Example: IBM i Access for Windows NLS	
cwbDQ_GetLibName . . . . .	134	APIs . . . . .	209
cwbDQ_GetQueueAttr . . . . .	134	IBM i Access for Windows Directory Update	
cwbDQ_GetQueueName . . . . .	135	APIs . . . . .	211
cwbDQ_GetSysName . . . . .	136	Typical use of IBM i Access for Windows	
cwbDQ_Peek . . . . .	136	Directory Update APIs . . . . .	212
cwbDQ_Read . . . . .	138	Requirements for Directory Update entries . . . . .	212
cwbDQ_Write . . . . .	139	Options for Directory Update entries . . . . .	213
Data Queues: Attributes APIs . . . . .	140	Directory Update package files syntax and	
cwbDQ_CreateAttr . . . . .	140	format . . . . .	214
cwbDQ_DeleteAttr . . . . .	140	Directory Update sample program . . . . .	214
cwbDQ_GetAuthority . . . . .	141	Directory Update: Create and delete APIs . . . . .	215
cwbDQ_GetDesc . . . . .	142	cwbUP_CreateUpdateEntry . . . . .	215
cwbDQ_GetForceToStorage . . . . .	142	cwbUP_DeleteEntry . . . . .	216
cwbDQ_GetKeySize . . . . .	143	Directory Update: Access APIs . . . . .	216
cwbDQ_GetMaxRecLen . . . . .	144	cwbUP_FindEntry . . . . .	217
cwbDQ_GetOrder . . . . .	144	cwbUP_FreeLock . . . . .	218
cwbDQ_GetSenderID . . . . .	145	cwbUP_GetEntryHandle . . . . .	218
cwbDQ_SetAuthority . . . . .	146	Directory Update: Free Resources APIs . . . . .	219
cwbDQ_SetDesc . . . . .	147	cwbUP_FreeEntryHandle . . . . .	219
cwbDQ_SetForceToStorage . . . . .	147	Directory Update: Change APIs . . . . .	220
cwbDQ_SetKeySize . . . . .	148	cwbUP_AddPackageFile . . . . .	220
cwbDQ_SetMaxRecLen . . . . .	149	cwbUP_RemovePackageFile . . . . .	221
cwbDQ_SetOrder . . . . .	149	cwbUP_SetCallbackDLL . . . . .	222

cwbUP_SetDescription . . . . .	222	Front Margin Offset Across . . . . .	242
cwbUP_SetEntryAttributes . . . . .	223	Front Margin Offset Down . . . . .	242
cwbUP_SetSourcePath . . . . .	224	Front Overlay Library Name . . . . .	242
cwbUP_SetTargetPath . . . . .	225	Front Overlay Name . . . . .	243
Directory Update: Information APIs . . . . .	226	Front Overlay Offset Across . . . . .	243
cwbUP_GetCallbackDLL . . . . .	226	Front Overlay Offset Down. . . . .	243
cwbUP_GetDescription . . . . .	227	Graphic Character Set . . . . .	243
cwbUP_GetEntryAttributes . . . . .	228	Hardware Justification . . . . .	243
cwbUP_GetLockHolderName . . . . .	228	Hold Spool File. . . . .	244
cwbUP_GetSourcePath . . . . .	229	Initialize the writer . . . . .	244
cwbUP_GetTargetPath . . . . .	230	Internet Address . . . . .	244
IBM i Access for Windows PC5250 emulation		Job Name . . . . .	244
APIs . . . . .	231	Job Number . . . . .	244
System Objects APIs for IBM i Access for		Job Separators . . . . .	245
Windows . . . . .	232	Job User . . . . .	245
System objects attributes. . . . .	233	Last Page Printed . . . . .	245
Advanced Function Printing . . . . .	233	Length of Page . . . . .	245
Align Page . . . . .	233	Library Name . . . . .	245
Allow Direct Print. . . . .	233	Lines Per Inch . . . . .	246
Authority. . . . .	233	Manufacturer Type and Model . . . . .	246
Authority to Check . . . . .	234	Maximum Spooled Output Records . . . . .	246
Automatically End Writer . . . . .	234	Measurement Method . . . . .	246
Back Margin Offset Across . . . . .	234	Message Help . . . . .	246
Back Margin Offset Down . . . . .	234	Message ID . . . . .	247
Backside Overlay Library Name . . . . .	234	Message Queue Library Name. . . . .	247
Backside Overlay Name . . . . .	235	Message Queue . . . . .	247
Back Overlay offset across . . . . .	235	Message Reply . . . . .	247
Back Overlay Offset Down . . . . .	235	Message Text . . . . .	248
Characters per Inch . . . . .	235	Message Type . . . . .	248
Code Page . . . . .	235	Message Severity . . . . .	248
Coded Font Name. . . . .	236	Number of Bytes to Read/Write . . . . .	248
Coded Font Library Name . . . . .	236	Number of Files . . . . .	248
Copies. . . . .	236	Number of Writers Started to Queue . . . . .	249
Copies left to Produce . . . . .	236	Object Extended Attribute . . . . .	249
Current page . . . . .	236	Open time commands . . . . .	249
Data Format. . . . .	237	Operator Controlled . . . . .	249
Data Queue Library Name . . . . .	237	Order of Files On Queue . . . . .	250
Data Queue Name. . . . .	237	Output Priority. . . . .	250
Date File Opened . . . . .	237	Output Queue Library Name . . . . .	250
User Specified DBCS Data . . . . .	237	Output Queue Name. . . . .	250
DBCS Extension Characters. . . . .	238	Output Queue Status. . . . .	250
DBCS Character Rotation . . . . .	238	Overflow Line Number . . . . .	251
DBCS Characters per Inch . . . . .	238	Pages Per Side . . . . .	251
DBCS SO/SI Spacing. . . . .	238	Pel Density . . . . .	251
Defer Write . . . . .	239	Point Size . . . . .	251
Degree of Page Rotation. . . . .	239	Print Fidelity . . . . .	251
Delete File After Sending . . . . .	239	Print on Both Sides . . . . .	252
Destination Option . . . . .	239	Print Quality . . . . .	252
Destination Type . . . . .	239	Print Sequence . . . . .	252
Device Class. . . . .	240	Print Text. . . . .	252
Device Model . . . . .	240	Printer. . . . .	252
Device Type . . . . .	240	Printer Device Type . . . . .	253
Display any File . . . . .	240	Printer File Library Name . . . . .	253
Drawer for Separators . . . . .	240	Printer File Name . . . . .	253
Ending Page. . . . .	241	Printer Queue . . . . .	253
File Separators . . . . .	241	Record Length . . . . .	253
Fold Records . . . . .	241	Remote System. . . . .	254
Font Identifier . . . . .	241	Replace Unprintable Characters . . . . .	254
Form Feed . . . . .	241	Replacement Character . . . . .	254
Form Type . . . . .	242	Resource library name . . . . .	254
Form Type Message Option . . . . .	242	Resource name . . . . .	254

Resource object type . . . . .	255	cwbOBJ_GetObjAttr . . . . .	279
Restart Printing. . . . .	255	cwbOBJ_GetObjAttrs . . . . .	283
Save Spooled File . . . . .	255	cwbOBJ_GetObjHandle . . . . .	284
Seek Offset . . . . .	255	cwbOBJ_GetObjHandleFromID . . . . .	285
Seek Origin . . . . .	255	cwbOBJ_GetObjID. . . . .	286
Send Priority . . . . .	256	cwbOBJ_RefreshObj . . . . .	287
Separator page . . . . .	256	cwbOBJ_SetObjAttrs . . . . .	288
Source Drawer . . . . .	256	IBM i Access for Windows Parameter object	
Spool SCS . . . . .	256	APIs . . . . .	291
Spool the Data . . . . .	256	cwbOBJ_CopyParmObjHandle. . . . .	291
Spooled File Name . . . . .	257	cwbOBJ_CreateParmObjHandle . . . . .	292
Spooled File Number. . . . .	257	cwbOBJ_DeleteParmObjHandle . . . . .	292
Spooled File Status . . . . .	257	cwbOBJ_GetParameter . . . . .	293
Spooled Output Schedule . . . . .	257	cwbOBJ_SetParameter . . . . .	294
Starting Page . . . . .	257	IBM i Access for Windows Writer job APIs	295
Text Description . . . . .	258	cwbOBJ_EndWriter . . . . .	295
Time File Opened . . . . .	258	cwbOBJ_StartWriter . . . . .	296
Total Pages . . . . .	258	IBM i Access for Windows output queues	
Transform SCS to ASCII . . . . .	258	APIs . . . . .	298
Unit of Measure . . . . .	258	cwbOBJ_HoldOutputQueue . . . . .	298
User Comment . . . . .	259	cwbOBJ_PurgeOutputQueue . . . . .	298
User Data . . . . .	259	cwbOBJ_ReleaseOutputQueue. . . . .	299
User defined data . . . . .	259	IBM i Access for Windows AFP resource	
User defined object library . . . . .	259	APIs . . . . .	300
User defined object name . . . . .	259	cwbOBJ_CloseResource . . . . .	300
User defined object type. . . . .	260	cwbOBJ_CreateResourceHandle . . . . .	301
User defined option(s) . . . . .	260	cwbOBJ_DisplayResource . . . . .	302
User driver program . . . . .	260	cwbOBJ_OpenResource . . . . .	303
User driver program library . . . . .	260	cwbOBJ_OpenResourceForSplF . . . . .	304
User driver program name . . . . .	260	cwbOBJ_ReadResource . . . . .	306
User ID . . . . .	261	cwbOBJ_SeekResource . . . . .	307
User ID Address . . . . .	261	IBM i Access for Windows APIs for new	
User transform program library . . . . .	261	spooled files. . . . .	308
User transform program name . . . . .	261	cwbOBJ_CloseNewSplF . . . . .	308
VM/MVS Class . . . . .	261	cwbOBJ_CloseNewSplFAndGetHandle	309
When to Automatically End Writer . . . . .	262	cwbOBJ_CreateNewSplF. . . . .	310
When to End Writer . . . . .	262	cwbOBJ_GetSplFHandleFromNewSplF	312
When to Hold File. . . . .	262	cwbOBJ_WriteNewSplF . . . . .	313
Width of Page . . . . .	262	APIs for reading spooled files for IBM i	
Workstation Customizing Object Name	262	Access for Windows . . . . .	314
Workstation Customizing Object Library	263	cwbOBJ_CloseSplF . . . . .	314
Writer Job Name . . . . .	263	cwbOBJ_OpenSplF . . . . .	315
Writer Job Number . . . . .	263	cwbOBJ_ReadSplF. . . . .	315
Writer Job Status . . . . .	263	cwbOBJ_SeekSplF . . . . .	317
Writer Job User Name . . . . .	263	APIs for manipulating spooled files for IBM i	
Writer Starting Page . . . . .	264	Access for Windows . . . . .	318
Network Print Server Object Attributes	264	cwbOBJ_CallExitPgmForSplF . . . . .	318
List APIs for IBM i Access for Windows . . . . .	266	cwbOBJ_CreateSplFHandle . . . . .	319
cwbOBJ_CloseList . . . . .	266	cwbOBJ_CreateSplFHandleEx . . . . .	320
cwbOBJ_CreateListHandle . . . . .	267	cwbOBJ_DeleteSplF . . . . .	321
cwbOBJ_DeleteListHandle . . . . .	268	cwbOBJ_DisplaySplF . . . . .	322
cwbOBJ_GetListSize . . . . .	269	cwbOBJ_HoldSplF. . . . .	323
cwbOBJ_OpenList . . . . .	270	cwbOBJ_IsViewerAvailable . . . . .	324
cwbOBJ_ResetListAttrsToRetrieve. . . . .	271	cwbOBJ_MoveSplF . . . . .	325
cwbOBJ_ResetListFilter . . . . .	271	cwbOBJ_ReleaseSplF . . . . .	326
cwbOBJ_SetListAttrsToRetrieve . . . . .	272	cwbOBJ_SendNetSplF . . . . .	326
cwbOBJ_SetListFilter . . . . .	273	cwbOBJ_SendTCPSplF . . . . .	328
cwbOBJ_SetListFilterWithSplF . . . . .	276	APIs for handling spooled file messages for	
IBM i Access for Windows Object APIs . . . . .	277	IBM i Access for Windows . . . . .	329
cwbOBJ_CopyObjHandle . . . . .	277	cwbOBJ_AnswerSplFMsg . . . . .	329
cwbOBJ_DeleteObjHandle . . . . .	278	cwbOBJ_GetSplFMsgAttr . . . . .	330



APIs for analyzing spooled file data for IBM i Access for Windows . . . . .	332	cwbSV_CreateTraceAPIHandle . . . . .	369
cwbOBJ_AnalyzeSplFData . . . . .	332	cwbSV_CreateTraceSPIHandle . . . . .	370
Server program APIs for IBM i Access for Windows . . . . .	333	cwbSV_DeleteTraceAPIHandle . . . . .	371
cwbOBJ_DropConnections . . . . .	333	cwbSV_DeleteTraceSPIHandle . . . . .	371
cwbOBJ_GetNPServerAttr . . . . .	333	cwbSV_LogAPIEntry . . . . .	372
cwbOBJ_SetConnectionsToKeep . . . . .	335	cwbSV_LogAPIExit . . . . .	373
Example: Using system objects APIs for IBM i Access for Windows . . . . .	336	cwbSV_LogSPIEntry . . . . .	374
IBM i Access for Windows Remote Command/Distributed Program Call APIs . . . . .	338	cwbSV_LogSPIExit . . . . .	375
Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs . . . . .	338	cwbSV_SetAPIComponent . . . . .	375
Remote Command/Distributed Program Call: Access remote command APIs list for IBM i Access for Windows . . . . .	340	cwbSV_SetAPIProduct . . . . .	376
cwbRC_GetClientCCSID . . . . .	340	cwbSV_SetSPIComponent . . . . .	377
cwbRC_GetHostCCSID . . . . .	341	cwbSV_SetSPIProduct . . . . .	378
cwbRC_StartSysEx . . . . .	342	Serviceability API list: Reading service files . . . . .	378
cwbRC_StopSys . . . . .	343	cwbSV_ClearServiceFile . . . . .	378
Remote Command/Distributed Program Call: Run APIs list for IBM i Access for Windows . . . . .	344	cwbSV_CloseServiceFile . . . . .	379
cwbRC_RunCmd . . . . .	344	cwbSV_CreateServiceRecHandle . . . . .	380
Remote Command/Distributed Program Call: Access programs APIs list for IBM i Access for Windows . . . . .	345	cwbSV_DeleteServiceRecHandle . . . . .	380
cwbRC_AddParm . . . . .	345	cwbSV_GetComponent . . . . .	381
cwbRC_CallPgm . . . . .	346	cwbSV_GetDateStamp . . . . .	382
cwbRC_CreatePgm . . . . .	347	cwbSV_GetMaxRecordSize . . . . .	383
cwbRC_DeletePgm . . . . .	349	cwbSV_GetMessageText . . . . .	384
cwbRC_GetLibName . . . . .	349	cwbSV_GetProduct . . . . .	385
cwbRC_GetParm . . . . .	350	cwbSV_GetRecordCount . . . . .	385
cwbRC_GetParmCount . . . . .	351	cwbSV_GetServiceFileName . . . . .	386
cwbRC_GetPgmName . . . . .	352	cwbSV_GetServiceType . . . . .	387
cwbRC_SetLibName . . . . .	353	cwbSV_GetTimeStamp . . . . .	388
cwbRC_SetParm . . . . .	354	cwbSV_GetTraceData . . . . .	389
cwbRC_SetPgmName . . . . .	355	cwbSV_GetTraceAPIData . . . . .	390
Example: Using Remote IBM i Access for Windows Command/Distributed Program Call APIs . . . . .	356	cwbSV_GetTraceAPIID . . . . .	391
IBM i Access for Windows Serviceability APIs . . . . .	358	cwbSV_GetTraceAPIType . . . . .	391
History log and trace files . . . . .	359	cwbSV_GetTraceSPIData . . . . .	392
Error handles . . . . .	360	cwbSV_GetTraceSPIID . . . . .	393
Typical use of Serviceability APIs . . . . .	360	cwbSV_GetTraceSPIType . . . . .	394
Serviceability APIs list: Writing to history log . . . . .	361	cwbSV_OpenServiceFile . . . . .	395
cwbSV_CreateMessageTextHandle . . . . .	361	cwbSV_ReadNewestRecord . . . . .	396
cwbSV_DeleteMessageTextHandle . . . . .	362	cwbSV_ReadNextRecord . . . . .	397
cwbSV_LogMessageText . . . . .	362	cwbSV_ReadOldestRecord . . . . .	397
cwbSV_SetMessageClass . . . . .	363	cwbSV_ReadPrevRecord . . . . .	398
cwbSV_SetMessageComponent . . . . .	364	Serviceability API list: Retrieving message text . . . . .	399
cwbSV_SetMessageProduct . . . . .	364	cwbSV_CreateErrHandle . . . . .	399
Serviceability APIs list: Writing trace data . . . . .	365	cwbSV_DeleteErrHandle . . . . .	400
cwbSV_CreateTraceDataHandle . . . . .	365	cwbSV_GetErrClass . . . . .	400
cwbSV_DeleteTraceDataHandle . . . . .	366	cwbSV_GetErrClassIndexed . . . . .	401
cwbSV_LogTraceData . . . . .	367	cwbSV_GetErrCount . . . . .	402
cwbSV_SetTraceComponent . . . . .	367	cwbSV_GetErrFileName . . . . .	403
cwbSV_SetTraceProduct . . . . .	368	cwbSV_GetErrFileNameIndexed . . . . .	404
Serviceability API list: Writing trace points . . . . .	369	cwbSV_GetErrLibName . . . . .	405
		cwbSV_GetErrLibNameIndexed . . . . .	406
		cwbSV_GetErrSubstText . . . . .	407
		cwbSV_GetErrSubstTextIndexed . . . . .	408
		cwbSV_GetErrText . . . . .	410
		cwbSV_GetErrTextIndexed . . . . .	411
		Example: Using IBM i Access for Windows Serviceability APIs . . . . .	412
		IBM i Access for Windows System Object Access (SOA) APIs . . . . .	413
		SOA objects . . . . .	414
		System object views . . . . .	414
		Typical use of System Object Access APIs for IBM i Access for Windows . . . . .	414

Display a customized list of system objects . . . . .	415	Establish ODBC connections . . . . .	483
Display the Properties view for a system object . . . . .	417	Execute ODBC functions . . . . .	484
Access and update data for system objects	419	Execute prepared statements . . . . .	485
IBM i Access for Windows System Object		ODBC API return codes . . . . .	489
Access programming considerations . . . . .	422	End ODBC functions . . . . .	489
About System Object Access errors . . . . .	422	Implementation issues of ODBC APIs . . . . .	490
System Object Access application profiles	423	ODBC 3.x API notes . . . . .	490
Manage IBM i communications sessions for application programs . . . . .	423	Connection string keywords . . . . .	497
System Object Access APIs for IBM i Access for Windows List . . . . .	424	Version and release changes in the ODBC driver behavior . . . . .	513
CWBSO_CloseList . . . . .	425	ODBC API restrictions and unsupported functions . . . . .	514
CWBSO_CopyObjHandle . . . . .	425	Signon dialog behavior . . . . .	514
CWBSO_CreateErrorHandle . . . . .	426	ODBC data types and how they correspond to DB2 for i database types . . . . .	515
CWBSO_CreateListHandle . . . . .	427	Working with the XML data type . . . . .	517
CWBSO_CreateListHandleEx . . . . .	428	Large objects (LOBs) considerations . . . . .	519
CWBSO_CreateObjHandle . . . . .	430	Connection and statement attributes . . . . .	520
CWBSO_CreateParmObjHandle . . . . .	431	Connection pooling . . . . .	523
CWBSO_DeleteErrorHandle . . . . .	431	SQLPrepare and SQLNativeSQL escape sequences and scalar functions . . . . .	523
CWBSO_DeleteListHandle . . . . .	432	Distributed transaction support . . . . .	524
CWBSO_DeleteObjHandle . . . . .	432	Cursor behavior notes . . . . .	524
CWBSO_DeleteParmObjHandle . . . . .	433	Extended dynamic disabled error . . . . .	525
CWBSO_DisallowListActions . . . . .	434	ODBC 64-bit Windows and Linux Considerations . . . . .	526
CWBSO_DisallowListFilter . . . . .	435	Restrictions of the 64-bit IBM i Access for Windows ODBC Driver . . . . .	529
CWBSO_DisplayErrMsg . . . . .	435	SQLTables Description . . . . .	529
CWBSO_DisplayList . . . . .	436	Handle long-running queries . . . . .	529
CWBSO_DisplayObjAttr . . . . .	437	Isolation level considerations . . . . .	530
CWBSO_GetErrMsgText . . . . .	438	IBM i Access for Windows ODBC performance . . . . .	530
CWBSO_GetListSize . . . . .	439	Performance-tuning IBM i Access for Windows ODBC . . . . .	530
CWBSO_GetObjAttr . . . . .	441	Performance considerations of common end-user tools . . . . .	534
CWBSO_GetObjHandle . . . . .	442	SQL performance . . . . .	536
CWBSO_OpenList . . . . .	443	ODBC support for multiple row statements . . . . .	544
CWBSO_ReadListProfile . . . . .	444	Catalog functions . . . . .	546
CWBSO_RefreshObj . . . . .	445	Exit programs . . . . .	546
CWBSO_ResetParmObj . . . . .	446	SQL and External procedures . . . . .	562
CWBSO_SetListFilter . . . . .	447	ODBC program examples . . . . .	571
CWBSO_SetListProfile . . . . .	448	Example: Visual C++ - Access and return data by calling a procedure . . . . .	571
CWBSO_SetListSortFields . . . . .	449	Example: Visual Basic - Access and return data by a call to a procedure . . . . .	573
CWBSO_SetListTitle . . . . .	450	Examples: ILE RPG - Host code for ODBC procedures . . . . .	575
CWBSO_SetObjAttr . . . . .	450	IBM i Access database APIs . . . . .	576
CWBSO_SetParameter . . . . .	451	Java programming . . . . .	576
CWBSO_WaitForObj . . . . .	452	ActiveX programming . . . . .	577
CWBSO_WriteListProfile . . . . .	453		
SOA attribute special values . . . . .	454		
IBM i Access for Windows: Database programming	468		
IBM i Access for Windows .NET provider . . . . .	468		
IBM i Access for Windows OLE DB provider	469		
IBM i Access ODBC . . . . .	470		
Files required to build an ODBC application	471		
Choose an interface to access the ODBC driver . . . . .	471		
ODBC C/C++ application header files . . . . .	472		
ODBC APIs: General concepts . . . . .	473		
Parameter markers . . . . .	473		
SQLFetch and SQLGetData . . . . .	474		
Code directly to ODBC APIs . . . . .	474		
Retrieve results . . . . .	480		
Access a database server with an ODBC application . . . . .	483		
		<b>Appendix. Notices . . . . .</b>	<b>579</b>
		Programming Interface Information . . . . .	581
		Trademarks . . . . .	581
		Terms and conditions . . . . .	581

---

## IBM i Access for Windows : Programming

As an application developer, explore this topic to reference and use IBM® i 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 IBM i resources. Using this topic, client/server applications are developed and tailored to the needs of your business. Various programming techniques are described so you can connect, manage, and take advantage of the rich functions provided by the server. You can access this information by selecting from the topics listed below.

- | If a basic working knowledge of IBM i Access for Windows features and functions is needed see **What's New** and the **User's Guide**, which are shipped with the IBM i Access for Windows product.

**Note:** To launch features from a Windows PC, select **Start** → **Programs** → **IBM i Access for Windows**, and select the component.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

---

### What's new for IBM i 7.1

This page highlights changes to the IBM i Access for Windows programming topic for IBM i 7.1.

The following are now supported by the .NET Data Provider:

- | • 128-byte schema names
- | • Support for the IBM i XML Data Type
- | • Connection property to configure Concurrent Access Resolution
- | • Support for multi-row UPDATE, DELETE, and MERGE statements
- | • Support Visual Studio 2008
- | • Online help now available in Visual Studio

The following are now supported by the OLE DB Data Provider:



- | • 128-byte schema names
- | • Support for the IBM i XML Data Type
- | • Connection property to configure Concurrent Access Resolution

The following are now supported by the ODBC driver:

- | • 128-byte schema names
- | • Support for the IBM i XML Data Type
- | • Connection property to configure Concurrent Access Resolution
- | • Support for multi-row UPDATE, DELETE, and MERGE statements

#### 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.

In PDF files, you might see revision bars (|) in the left margin of new and changed information.

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

---

## PDF file for IBM i Access for Windows: Programming

You can view and print a PDF file of this information.


To view or download the PDF version of this document, select IBM i Access for Windows Programming (about 1890 KB).

### Saving PDF files

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

1. Right-click the PDF in your browser (right-click the link above).
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

### Downloading Adobe® Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site ([get.adobe.com/reader/](http://get.adobe.com/reader/)) .

---

## IBM i Access for Windows C/C++ APIs

IBM i Access for Windows C/C++ application programming interfaces (APIs) are used to access IBM i resources.

These APIs are intended primarily for C/C++ programmers. They are also called from other languages that support calling C-style APIs.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

### IBM i Access for Windows C/C++ APIs overview

See the following topics for IBM i Access for Windows C/C++ APIs overview information.

#### API groups, header files, import libraries, and DLLs

Access interface definition files for all IBM i Access for Windows C/C++ API groups in the IBM i Access for Windows **Programmer's Toolkit**.

For each IBM i 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

#### How to access IBM i Access for Windows header files in the Toolkit:

1. Find the **Programmer's Toolkit** icon in your IBM i 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 IBM i Access for Windows programming:

To find this IBM i 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
IBM i Object	IBM i 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 IBM i 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 IBM i 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 cwbdq.lib %windir%\system32\cwbdq.dll
```

*Table 1. IBM i 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
Data Queues	cwbdq.h	cwbapi.lib	cwbdq.dll
Data transformation	cwbdh.h	cwbapi.lib	cwbdh.dll
Directory Update	cwbup.h	cwbapi.lib	cwbup.dll
Emulation (Standard HLLAPI interface)	hapi_c.h	pscal32.lib	pcshll.dll pcshll32.dll
Emulation (Enhanced HLLAPI interface)	ehlapi32.h	ehlapi32.lib	ehlapi32.dll
Emulation (Windows EHLLAPI interface)	whllapi.h	whllapi.lib	whllapi.dll
		whlapi32.lib	whlapi32.dll
Emulation (HAEL interface)	eclall.hpp	pcseclva.lib	pcseclva.dll
		pcseclvc.lib	pcseclvc.dll
Emulation (PCSAPI interface)	pcsapi.h	pscal32.lib	pcsapi.dll pcsapi32.dll

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

API group	Header file	Import library	DLL
National language support (General NLS)	cwbnl.h	cwbapi.lib	cwbnl.dll
National language support (Conversion NLS)	cwbnlcnv.h	cwbapi.lib	cwbcore.dll
National language support (Dialog-box NLS)	cwbnldlg.h	cwbapi.lib	cwbnldlg.dll
IBM i objects	cwbobj.h	cwbapi.lib	cwbobj.dll
ODBC	sql.h sqlext.h sqltypes.h sqlucode.h	odbc32.lib	odbc32.dll
Database APIs (Optimized SQL) <b>Note:</b> These APIs are no longer being enhanced.	cwbdb.h	cwbapi.lib	cwbdb.dll
OLE DB Provider	ad400.h da400.h		cwbzzodb.dll  See the OLE DB Section of the Microsoft 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	cwbsoapi.h	cwbapi.lib	cwbsoapi.dll

### Related reference

“OEM, ANSI, and Unicode considerations” on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

“Use a single IBM i Access for Windows API type” on page 8

To restrict your application to a particular type of IBM i Access for Windows API, you must define one, and only one preprocessor definition.

### Programmer’s Toolkit

Find header files and complete information to develop IBM i Access for Windows applications.

The IBM i Access for Windows Programmer’s Toolkit is an installable component of the IBM i Access for Windows product and is the primary source of information needed to develop IBM i Access for Windows applications. This includes programming with IBM i Access for Windows ActiveX Automation Objects, ADO/OLE DB, .NET, and Java™. The Programmer’s Toolkit contains links to header files, sample programs, and complete documentation.

#### Notes:

- No portion of the Toolkit or the IBM i Access for Windows product may be redistributed with the resulting applications.
- By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578

The Programmer's Toolkit consists of two parts:

- The IBM i Access for Windows Programmer's Toolkit component, which includes:
  - Online help information for the Toolkit and other online help for the product.
  - C/C++ header files
  - C import libraries
  - ActiveX automation type libraries
- Programmer's Toolkit Web site which includes sample applications and tools that are useful for developing IBM i Access for Windows applications. The site is updated regularly. Check it periodically for new information.

#### Related information

 [IBM i Access for Windows Toolkit](#)

#### Install the Programmer's Toolkit:

The Programmer's Toolkit is installed as a feature of the IBM i Access for Windows product.

To add or remove the Programmer's Toolkit and other features of the product, use the Add or Remove Programs in your PC Control Panel.

1. Select **Start** → **Control Panel** → **Add or Remove Programs** → **IBM i Access for Windows** → **Change**
2. Follow the instructions on the screen, selecting the Modify button.
3. Click the feature name (Programmer's Toolkit) and choose one of these, as appropriate:
  - **This feature will be installed on local hard drive.** (To install a feature)
  - **This feature, and all subfeatures, will be installed on local hard drive.** (To install features.)
  - **This feature will not be available.** (To remove a feature.)
4. Click Install to modify the features that are installed and continue through the Install wizard until it completes.

#### Related reference

"ActiveX programming" on page 577

ActiveX automation is a programming technology that is defined by Microsoft and is supported by the IBM i Access for Windows product.

#### Launch the Programmer's Toolkit:

The Programmer's Toolkit is launched as a feature of the IBM i Access for Windows product.

1. Install the Programmer's Toolkit feature on your personal computer.
2. Select **Start** → **Programs** → **IBM i Access for Windows** → **Programmer's Toolkit**

**Note:** The Toolkit icon appears only after you have installed the Programmer's Toolkit on your personal computer.

#### Related reference

"ActiveX programming" on page 577

ActiveX automation is a programming technology that is defined by Microsoft and is supported by the IBM i Access for Windows product.

#### IBM i name formats for connection APIs

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

The valid formats are:

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

### Related reference

“IBM i Access for Windows Administration APIs” on page 32

These APIs provide functions that access information about the IBM i Access for Windows code that is installed on the PC.

“IBM i Access for Windows Communications and Security APIs” on page 43

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

“IBM i Data Queues APIs” on page 121

Use IBM i Access for Windows Data Queues application programming interfaces (APIs) to provide easy access to IBM i data queues. Data queues allow you to create client/server applications that do not require the use of communications APIs.

“IBM i Access for Windows data transformation APIs” on page 165

IBM i Access for Windows **data transformation** application programming interfaces (APIs) enable your client/server applications to transform IBM i numeric data between the system and the PC formats. Transformation may be required when you send and receive IBM i numeric data to and from the system. Data transformation APIs support transformation of many numeric formats.

“IBM i Access for Windows National Language Support (NLS) APIs” on page 183

National Language Support APIs enable your applications to get and save (query and change) the IBM i Access for Windows settings that are relevant to different language versions.

“IBM i Access for Windows Directory Update APIs” on page 211

Specify PC directory updates using the IBM i Access for Windows Directory Update function.

“System Objects APIs for IBM i Access for Windows” on page 232

System objects for IBM i Access for Windows application programming interfaces (APIs) allow you to work with print-related objects that are on the system. These APIs make it possible to work with IBM i spooled files, writer jobs, output queues, printers, and more.

“IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

The IBM i Access for Windows Remote Command/Distributed Program Call APIs allow the PC application programmer to access IBM i functions. User program and system commands are called without requiring an emulation session. A single IBM i program serves commands and programs, so only one system job is started for both.

“IBM i Access for Windows System Object Access (SOA) APIs” on page 413

System Object Access enables you to view and manipulate system objects through a graphical user interface.

### OEM, ANSI, and Unicode considerations

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

The generic version of the IBM i 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
```

Table 2. API types, name formats, and pre-processor definition

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
```

Select the related link to 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.

## Related reference

“IBM i Access for Windows Administration APIs” on page 32

These APIs provide functions that access information about the IBM i Access for Windows code that is installed on the PC.

“IBM i Access for Windows Communications and Security APIs” on page 43

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

“IBM i Data Queues APIs” on page 121

Use IBM i Access for Windows Data Queues application programming interfaces (APIs) to provide easy access to IBM i data queues. Data queues allow you to create client/server applications that do not require the use of communications APIs.

“IBM i Access for Windows data transformation APIs” on page 165

IBM i Access for Windows **data transformation** application programming interfaces (APIs) enable your client/server applications to transform IBM i numeric data between the system and the PC formats. Transformation may be required when you send and receive IBM i numeric data to and from the system. Data transformation APIs support transformation of many numeric formats.

“IBM i Access for Windows National Language Support (NLS) APIs” on page 183

National Language Support APIs enable your applications to get and save (query and change) the IBM i Access for Windows settings that are relevant to different language versions.

“IBM i Access for Windows Directory Update APIs” on page 211

Specify PC directory updates using the IBM i Access for Windows Directory Update function.

“System Objects APIs for IBM i Access for Windows” on page 232

System objects for IBM i Access for Windows application programming interfaces (APIs) allow you to work with print-related objects that are on the system. These APIs make it possible to work with IBM i spooled files, writer jobs, output queues, printers, and more.

“IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

The IBM i Access for Windows Remote Command/Distributed Program Call APIs allow the PC application programmer to access IBM i functions. User program and system commands are called without requiring an emulation session. A single IBM i program serves commands and programs, so only one system job is started for both.

“API groups, header files, import libraries, and DLLs” on page 2

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

## Use a single IBM i Access for Windows API type:

To restrict your application to a particular type of IBM i Access for Windows API, you must define one, and only one preprocessor definition.

The preprocessor definition is as follows:

- 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 the preprocessor definition and API names. Select the related link below for the API groups, header files, import libraries, and DLLs topic collection for more information.

## Related reference

“API groups, header files, import libraries, and DLLs” on page 2  
Access interface definition files for all IBM i Access for Windows C/C++ API groups in the IBM i Access for Windows **Programmer’s Toolkit**.

## Use mixed IBM i Access for Windows API types:

You can mix ANSI, OEM, and Unicode APIs by using explicit IBM i Access for Windows API names.

For example, you can write an ANSI IBM i Access for Windows application by specifying the CWB\_ANSI preprocessor definition, but still call a Unicode version of an API by using the “W” suffix.

## Write a generic IBM i Access for Windows application:

Generic IBM i Access for Windows applications allow maximum portability because the same source code can be compiled for OEM, ANSI, and Unicode.

Generic applications are built by specifying different preprocessor definitions, and by using the generic version of the IBM i 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 preprocessor definitions for compilation. When compiling your source for Unicode in Visual C++, you should also use the preprocessor 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 (See related links.)
4. MSDN Library (See related links.)

## Related information

 Knowledge Base

MSDN Library

## Obsolete IBM i Access for Windows APIs

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

The APIs for the following four functions are obsolete and IBM i Access for Windows support is no longer provided.

- **APPC**
- **License Management**
- **Ultimedia System Facilities (USF)**
- **Messaging Application Programming Interface (MAPI)**

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

## **Obsolete IBM i Access for Windows APIs list:**

### **Obsolete Communications APIs:**

There are some IBM i Access for Windows obsolete communications APIs.

#### **cwbCO\_IsSystemConfigured**

IBM i Access for Windows function does not require pre-configuration to use an IBM i connection. For this reason, programs that need an IBM i connection (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 55.

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

#### **cwbCO\_GetUserID**

Use “`cwbCO_GetUserIDEx`” on page 66.

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

#### **cwbCO\_GetHostVersion**

Use “`cwbCO_GetHostVersionEx`” on page 85.

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

### **Obsolete Data Queues APIs:**

There are some IBM i Access for Windows obsolete data queue APIs.

#### **cwbDQ\_Create**

Use “`cwbDQ_CreateEx`” on page 124

#### **cwbDQ\_Delete**

Use “`cwbDQ_DeleteEx`” on page 126

#### **cwbDQ\_Open**

Use “`cwbDQ_OpenEx`” on page 127

**cwbDQ\_StartSystem**

Use "cwbCO\_Connect" on page 50

**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 50 for details.

**cwbDQ\_StopSystem**

Use "cwbCO\_Disconnect" on page 52

**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 52 for details.

**Obsolete Remote Command/Distributed Program Call APIs:**

There are some IBM i Access for Windows obsolete Remote Command and distributed program call APIs.

**cwbRC\_StartSys**

Use "cwbRC\_StartSysEx" on page 342

**cwbRC\_GetSysName**

Use "cwbCO\_GetSystemName" on page 88

**Obsolete Security APIs:**

There are some IBM i Access for Windows obsolete security APIs.

**cwbSY\_CreateSecurityObj**

Use "cwbCO\_CreateSystem" on page 47

**cwbSY\_DeleteSecurityObj**

Use "cwbCO\_DeleteSystem" on page 50

**cwbSY\_SetSys**

Use "cwbCO\_CreateSystem" on page 47 and pass a system name on the call

**cwbSY\_VerifyUserIDPwd**

Use "cwbCO\_VerifyUserIDPassword" on page 77

**cwbSY\_ChangePwd**

Use "cwbCO\_ChangePassword" on page 59

**cwbSY\_GetUserID**

Use "cwbCO\_GetUserIDEx" on page 66

**cwbSY\_Logon**

Use "cwbCO\_Signon" on page 76

**cwbSY\_LogonUser**

Use "cwbCO\_SetUserIDEx" on page 73, "cwbCO\_SetPassword" on page 71, or "cwbCO\_Signon" on page 76

**cwbSY\_GetDateTimeCurrentSignon**

Use "cwbCO\_GetSignonDate" on page 65

**cwbSY\_GetDateTimeLastSignon**

Use "cwbCO\_GetPrevSignonDate" on page 63

**cwbSY\_GetDateTimePwdExpires**

Use "cwbCO\_GetPasswordExpireDate" on page 62

**cwbSY\_GetFailedAttempts**

Use "cwbCO\_GetFailedSignons" on page 61

### Obsolete Serviceability APIs:

There are some IBM i Access for Windows 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:

There are some IBM i Access for Windows obsolete SOA APIs.

**CWBSO\_CreateListHandle**

Use "CWBSO\_CreateListHandleEx" on page 428

### Obsolete National Language Support (NLS) APIs:

There are some IBM i Access for Windows obsolete NLS APIs.

## **cwbNL\_CreateConverter**

Use "cwbNL\_CreateConverterEx" on page 198

## **cwbNL\_ConvertCodePages**

Use "cwbNL\_ConvertCodePagesEx" on page 194

## **Return codes and error messages**

The IBM i 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.

IBM i Access for Windows error messages are logged in the History Log, and also on the 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 IBM i Access for Windows User's Guide, which is shipped with product, for information on starting the History Log

#### **Viewing logged messages:**

To view messages that have been logged in the History Log, select **Start** → **Programs** → **IBM i 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 IBM i Access for Windows User's Guide.

### **IBM i error messages:**

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

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

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

### **IBM i Access for Windows return codes that correspond to operating system errors:**

There is a relationship between IBM i Access for Windows return codes and system error messages.

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.

### **IBM i Access for Windows return codes:**

There are global and specific IBM i Access for Windows return codes.

*Global IBM i Access for Windows return codes:*

There are global IBM i Access for Windows 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  
The server is inactive or does not exist.

4013 CWB\_NOT\_COMPATIBLE  
IBM i Access program or function not at correct level.

4014 CWB\_INVALID\_POINTER  
A pointer is NULL.

4015 CWB\_SERVER\_PROGRAM\_NOT\_FOUND  
The server application not found.

4016 CWB\_API\_ERROR  
General API failure.

4017 CWB\_CA\_NOT\_STARTED  
IBM i Access program 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 the IBM i Access function 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.

### Related reference

“IBM i Access for Windows Communications and Security APIs” on page 43  
The IBM i Access for Windows Communications and Security topic shows you how to use IBM i Access for Windows application programming interfaces (APIs)

*IBM i Access for Windows-specific return codes:*

There are specific IBM i Access for Windows return codes.

*Security return codes:*

There are IBM i Access for Windows 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
8027   CWB_EXIT_PGM_DENIED_REQUEST
8050   CWB_TIMESTAMPS_NOT_SET
8051   CWB_KERB_CLIENT_CREDENTIALS_NOT_FOUND
8052   CWB_KERB_SERVICE_TICKET_NOT_FOUND
8053   CWB_KERB_SERVER_CANNOT_BE_CONTACTED
8054   CWB_KERB_UNSUPPORTED_BY_HOST
8055   CWB_KERB_NOT_AVAILABLE
8056   CWB_KERB_SERVER_NOT_CONFIGURED
8057   CWB_KERB_CREDENTIALS_NOT_VALID
8058   CWB_KERB_MAPPED_USERID_FAILURE
8059   CWB_KERB_MAPPED_USERID_SUCCESS
8070   CWB_PROFILE_TOKEN_INVALID
8071   CWB_PROFILE_TOKEN_MAXIMUM
8072   CWB_PROFILE_TOKEN_NOT_REGENERABLE
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
8271   CWB_PW_STAR_NONE
8272   CWB_PW_QPWDVLDPGM
```

*Communications return codes:*

There are IBM i Access for Windows communications return codes.

```
8400   CWB_INV_AFTER_SIGNON
8401   CWB_INV_WHEN_CONNECTED
8402   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_CCSID_USED
8411   CWB_USER_TIMEOUT
8412   CWB_SSL_JAVA_ERROR
8413   CWB_USER_TIMEOUT_SENDRCV
8414   CWB_FIPS_UNAVAILABLE
```

*Configuration return codes:*

There are IBM i Access for Windows 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:*

There are IBM i Access for Windows 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:*

There are IBM i Access for Windows 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 system name was not found during TCP/IP
      address lookup.
11002  CWB_HOST_NOT_FOUND_BUT_WE_ARE_NOT_SURE
      The system name was not found during TCP/IP
      address lookup.
11004  CWB_VALID_NAME_BUT_NO_DATA_RECORD
      The service name was not found in the local
      SERVICES file.
```

*SSL return codes:*

There are IBM i Access for Windows SSL return codes.

## Key Database error codes

- 20001 - An unknown error occurred.
- 20002 - An asn.1 encoding/decoding error occurred.
- 20003 - An error occurred while initializing asn.1 encoder/decoder.
- 20004 - An asn.1 encoding/decoding error occurred because of an out-of-range index or nonexistent optional field.
- 20005 - A database error occurred.
- 20006 - An error occurred while opening the database file.
- 20007 - An error occurred while re-opening the database file.
- 20008 - Database creation failed.
- 20009 - The database already exists.
- 20010 - An error occurred while deleting the database file.
- 20011 - Database has not been opened.
- 20012 - An error occurred while reading the database file.
- 20013 - An error occurred while writing data to the database file.
- 20014 - A database validation error occurred.
- 20015 - An invalid database version was encountered.
- 20016 - An invalid database password was encountered.
- 20017 - An invalid database file type was encountered.
- 20018 - The database has been corrupted.
- 20019 - An invalid password was encountered or the database is not valid.
- 20020 - A database key entry integrity error occurred.
- 20021 - A duplicate key already exists in the database.
- 20022 - A duplicate key already exists in the database (Record ID).
- 20023 - A duplicate key already exists in the database (Label).
- 20024 - A duplicate key already exists in the database (Signature).
- 20025 - A duplicate key already exists in the database (Unsigned Certificate).
- 20026 - A duplicate key already exists in the database (Issuer and Serial Number).
- 20027 - A duplicate key already exists in the database (Subject Public Key Info).
- 20028 - A duplicate key already exists in the database (Unsigned CRL).
- 20029 - The label has been used in the database.
- 20030 - A password encryption error occurred.
- 20031 - An LDAP related error occurred.
- 20032 - A cryptographic error occurred.
- 20033 - An encryption/decryption error occurred.
- 20034 - An invalid cryptographic algorithm was found.
- 20035 - An error occurred while signing data.
- 20036 - An error occurred while verifying data.
- 20037 - An error occurred while computing digest of data.
- 20038 - An invalid cryptographic parameter was found.
- 20039 - An unsupported cryptographic algorithm was encountered.
- 20040 - The specified input size is greater than the supported modulus size.
- 20041 - An unsupported modulus size was found
- 20042 - A database validation error occurred.
- 20043 - Key entry validation failed.
- 20044 - A duplicate extension field exists.
- 20045 - The version of the key is wrong
- 20046 - A required extension field does not exist.
- 20047 - The validity period does not include today or does not fall within its issuer's validity period.
- 20048 - The validity period does not include today or does not fall within its issuer's validity period.
- 20049 - An error occurred while validating validity private key usage extension.
- 20050 - The issuer of the key was not found.
- 20051 - A required certificate extension is missing.
- 20052 - The key signature validation failed.
- 20053 - The key signature validation failed.
- 20054 - The root key of the key is not trusted.
- 20055 - The key has been revoked.
- 20056 - An error occurred while validating authority key identifier extension.
- 20057 - An error occurred while validating private key usage extension.
- 20058 - An error occurred while validating subject alternative name extension.
- 20059 - An error occurred while validating issuer alternative name extension.
- 20060 - An error occurred while validating key usage extension.
- 20061 - An unknown critical extension was found.
- 20062 - An error occurred while validating key pair entries.

20063 - An error occurred while validating CRL.  
20064 - A mutex error occurred.  
20065 - An invalid parameter was found.  
20066 - A null parameter or memory allocation error was encountered.  
20067 - Number or size is too large or too small.  
20068 - The old password is invalid.  
20069 - The new password is invalid.  
20070 - The password has expired.  
20071 - A thread related error occurred.  
20072 - An error occurred while creating threads.  
20073 - An error occurred while a thread was waiting to exit.  
20074 - An I/O error occurred.  
20075 - An error occurred while loading CMS.  
20076 - A cryptography hardware related error occurred.  
20077 - The library initialization routine was not successfully called.  
20078 - The internal database handle table is corrupted.  
20079 - A memory allocation error occurred.  
20080 - An unrecognized option was found.  
20081 - An error occurred while getting time information.  
20082 - Mutex creation error occurred.  
20083 - An error occurred while opening message catalog.  
20084 - An error occurred while opening error message catalog.  
20085 - An null file name was found.  
20086 - An error occurred while opening files, check for file existence and permissions.  
20087 - An error occurred while opening files to read.  
20088 - An error occurred while opening files to write.  
20089 - There is no such file.  
20090 - The file cannot be opened because of its permission setting.  
20091 - An error occurred while writing data to files.  
20092 - An error occurred while deleting files.  
20093 - Invalid Base64-encoded data was found.  
20094 - An invalid Base64 message type was found.  
20095 - An error occurred while encoding data with Base64 encoding rule.  
20096 - An error occurred while decoding Base64-encoded data.  
20097 - An error occurred while getting a distinguished name tag.  
20098 - The required common name field is empty.  
20099 - The required country name field is empty.  
20100 - An invalid database handle was found.  
20101 - The key database does not exist.  
20102 - The request key pair database does not exist.  
20103 - The password file does not exist.  
20104 - The new password is identical to the old one.  
20105 - No key was found in the key database.  
20106 - No request key was found.  
20107 - No trusted CA was found  
20108 - No request key was found for the certificate.  
20109 - There is no private key in the key database  
20110 - There is no default key in the key database.  
20111 - There is no private key in the key record.  
20112 - There is no certificate in the key record.  
20113 - There is no CRL entry.  
20114 - An invalid key database file name was found.  
20115 - An unrecognized private key type was found.  
20116 - An invalid distinguished name input was found.  
20117 - No key entry was found that has the specified key label.  
20118 - The key label list has been corrupted.  
20119 - The input data is not valid PKCS12 data.  
20120 - The password is invalid or the PKCS12 data has been corrupted or been created with later version of PKCS12.  
20121 - An unrecognized key export type was found.  
20122 - An unsupported password-based encryption algorithm was found.  
20123 - An error occurred while converting the keyring file to a CMS key database.  
20124 - An error occurred while converting the CMS key database to a keyring file.  
20125 - An error occurred while creating a certificate for the certificate request.  
20126 - A complete issuer chain cannot be built.  
20127 - Invalid WEBDB data was found.  
20128 - There is no data to be written to the keyring file.

- 20129 - The number of days that you entered extends beyond the permitted validity period.
- 20130 - The password is too short; it must consist of at least characters.
- 20131 - A password must contain at least one numeric digit.
- 20132 - All characters in the password are either alphabetic or numeric characters.
- 20133 - An unrecognized or unsupported signature algorithm was specified.
- 20134 - An invalid key database type was specified.
- 20135 - The secondary key database is currently a secondary key database to another primary key database.
- 20136 - The key database does not have a secondary key database associated with it.
- 20137 - A cryptographic token with label cannot be found.
- 20138 - A cryptographic token password was not specified but is required.
- 20139 - A cryptographic token password was specified but is not required.
- 20140 - The cryptographic module cannot be loaded. Cryptographic token support will not be available.
- 20141 - The function is not supported for cryptographic tokens.
- 20142 - The cryptographic token function failed.

#### SSL error codes

- 25001 - The handle is not valid.
- 25002 - The dynamic link library is not available.
- 25003 - An internal error occurred.
- 25004 - Main memory is insufficient to perform the operation.
- 25005 - The handle is not in a valid state for operation.
- 25006 - The key label is not found.
- 25007 - The certificate is not available.
- 25008 - Certificate validation error.
- 25009 - Error processing cryptography.
- 25010 - Error validating ASN fields in certificate.
- 25011 - Error connecting to LDAP server.
- 25012 - Internal unknown error. Report problem to service.
- 25101 - An error occurred processing the cipher.
- 25102 - I/O error reading key file.
- 25103 - Key file has an invalid internal format. Re-create key file.
- 25104 - Key file has two entries with the same key. Use iKeyman to remove the duplicate key.
- 25105 - Key file has two entries with the same label. Use iKeyman to remove the duplicate label.
- 25106 - The key file password is used as an integrity check. Either the key file has become corrupted or the password ID is incorrect.
- 25107 - The default key in the key file has an expired certificate. Use iKeyman to remove certificates that are expired.
- 25108 - There was an error loading one of the dynamic link libraries.
- 25109 - A connection is trying to be made after environment has been closed.
- 25201 - The key file could not be initialized.
- 25202 - Unable to open the key file. Either the path was specified incorrectly or the file permissions did not allow the file to be opened.
- 25203 - Unable to generate a temporary key pair.
- 25204 - A User Name object was specified that is not found.
- 25205 - A Password used for an LDAP query is not correct.
- 25206 - An index into the Fail Over list of LDAP servers was not correct.
- 25301 - An error occurred on close.
- 25401 - The system date was set to an invalid value.
- 25402 - Neither SSLV2 nor SSLV3 is enabled.
- 25403 - The required certificate was not received from partner.
- 25404 - The received certificate was formatted incorrectly.
- 25405 - The received certificate type was not supported.
- 25406 - An IO error occurred on a data read or write.
- 25407 - The specified label in the key file could not be found.
- 25408 - The specified key-file password is incorrect. The key file could not be used. The key file may also be corrupt.
- 25409 - In a restricted cryptography environment, the key size is too long to be supported.
- 25410 - An incorrectly formatted SSL message was received from the partner.
- 25411 - The message authentication code (MAC) was not successfully verified.
- 25412 - The operation is unsupported.
- 25413 - The received certificate contained an incorrect signature.
- 25414 - The server certificate is not trusted. This usually occurs when you have not downloaded the certificate authority for the server certificate. Use the

Digital Certificate Manager to obtain the certificate authority and use the PC IBM Key Management utility to place the certificate authority in your local key database. See CWBC01050 for additional information

- 25415 - The remote system information is not valid.
- 25416 - Access denied.
- 25417 - The self-signed certificate is not valid.
- 25418 - The read failed.
- 25419 - The write failed.
- 25420 - The partner closed the socket before the protocol completed. This could mean the partner is configured for SSL Client Authentication and no client certificate was sent to the partner.
- 25421 - The specified V2 cipher is not valid.
- 25422 - The specified V3 cipher is not valid.
- 25425 - The handle could not be created.
- 25426 - Initialization failed.
- 25427 - When validating a certificate, unable to access the specified LDAP directory.
- 25428 - The specified key did not contain a private key.
- 25429 - A failed attempt was made to load the specified PKCS11 shared library.
- 25430 - The PKCS #11 driver failed to find the token specified by the caller.
- 25431 - The PKCS #11 token is not present in the slot.
- 25432 - The password/pin to access the PKCS #11 token is invalid.
- 25433 - The SSL header received was not a properly SSLV2 formatted header.
- 25434 - Unable to access the hardware-based cryptographic service provider (CSP).
- 25435 - Attribute setting conflict
- 25436 - The requested function is not supported on the platform that the application is running
- 25437 - An IPv6 connection is detected
- 25438 - Incorrect value is returned from the reset session type callback function
- 25501 - The buffer size is negative or 0.
- 25502 - Used with non-blocking I/O.
- 25601 - SSLV3 is required for reset\_cipher, and the connection uses SSLV2.
- 25602 - An invalid ID was specified for the function call.
- 25701 - The function call has an invalid ID.
- 25702 - The attribute has a negative length, which is invalid.
- 25703 - The enumeration value is invalid for the specified enumeration type.
- 25704 - Invalid parameter list for replacing the SID cache routines.
- 25705 - When setting a numeric attribute, the specified value is invalid for the specific attribute being set.
- 25706 - Conflicting parameters have been set for additional certificate validation.
- 25707 - The cipher spec included an AES cipher spec that is not supported on the system of execution.
- 25708 - The length of the peer ID is incorrect. It must be less than or equal to 16 bytes

### **IBM i Access for Windows component-specific return codes:**

There are IBM i Access for Windows return codes for the API type.

#### *Administration APIs return code:*

There is an IBM i Access for Windows administration return code.

6001 CWBAD\_INVALID\_COMPONENT\_ID  
The component ID is invalid.

#### **Related reference**

“IBM i Access for Windows Administration APIs” on page 32

These APIs provide functions that access information about the IBM i Access for Windows code that is installed on the PC.

#### *Communications APIs return codes:*

There are IBM i Access for Windows communications API 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.

**Related reference**

“IBM i Access for Windows Communications and Security APIs” on page 43  
 The IBM i Access for Windows Communications and Security topic shows you how to use IBM i Access for Windows application programming interfaces (APIs)

*Database APIs return codes:*

There are IBM i Access for Windows database APIs return codes.

**Note:** See the IBM i Access for Windows database APIs topic for important information regarding database APIs.

- 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 system 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  
The data stream received from the host was too short.

6035 CWBDB\_SQL\_WARNING  
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

6048 CWBDB\_GENERAL\_SERVER\_ERROR The database server returned a general warning.  
 6049 CWBDB\_EXIT\_PROGRAM\_WARNING The database server returned a general error.  
 6050 CWBDB\_EXIT\_PROGRAM\_ERROR The database server returned a warning from an exit program.  
 6051 CWBDB\_DATA\_BUFFER\_TOO\_SMALL The database server returned an error from an exit program.  
 6052 CWBDB\_NL\_CONVERSION\_ERROR Target data buffer is smaller than source buffer.  
 6053 CWBDB\_COMMUNICATIONS\_ERROR Received error back from PiNConverter.  
 6054 CWBDB\_INVALID\_ARG\_API Received a communications error during processing.  
 6055 CWBDB\_MISSING\_DATA\_HANDLER Value of argument is invalid - API level.  
 6056 CWBDB\_REQUEST\_DATASTREAM\_NOT\_VALID Data handler not found in data handler list.  
 6057 CWBDB\_SERVER\_UNABLE Invalid datastream in catalog request.  
 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 Unable to start server because of set client level problem.  
 6063 CWBDB\_CLIENT\_LFC\_ERROR 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.

6079 CWBDB\_SERVER\_REL\_DB\_CONVERSION\_ERROR  
Unable to start server because of data conversion error on server relational DB field of server attributes. There is no message or help text for this return code.

6080 CWBDB\_SERVER\_FUNCTION\_NOT\_AVAILABLE  
This function is not available on this version of the host server.

6081 CWBDB\_FUNCTION\_NOT\_VALID\_AFTER\_CONNECT  
This function is not valid after connecting to the host server.

6082 CWBDB\_INVALID\_INITIAL\_REL\_DB\_NAME  
The initial relational DB name (IASP) was invalid.

6099 CWBDB\_LAST\_STREAM\_CHUNK  
Stream fetch complete.  
NOTE: Informational, not an error. There is not a message or help text for this return code.

### Related reference

“IBM i Access database APIs” on page 576

Use other technologies for functions that were provided by the IBM i Access for Windows proprietary C/C++ Database APIs, that are no longer being enhanced.

### *Data Queues APIs return codes:*

There are IBM i Access for Windows data queues API 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.

6014 CWBDQ\_INVALID\_TIME  
Wait time is not correct.

6015 CWBDQ\_INVALID\_SEARCH

6016 CWBDQ\_DATA\_TRUNCATED  
Search order is not correct.  
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  
system 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  
system object syntax. First character must be alphabetic and all  
following characters alphanumeric.

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.

### Related reference

“IBM i Data Queues APIs” on page 121

Use IBM i Access for Windows Data Queues application programming interfaces (APIs) to provide easy access to IBM i data queues. Data queues allow you to create client/server applications that do not require the use of communications APIs.

*Directory Update APIs return codes:*

There are IBM i Access for Windows Directory Update API 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

- 6004      Position that is given is not in range.  
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.

**Related reference**

“IBM i Access for Windows Directory Update APIs” on page 211  
Specify PC directory updates using the IBM i Access for Windows Directory Update function.

*National Language Support APIs return codes:*

There are IBM i Access for Windows NLS API 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

```

*System Object APIs return codes:*

There are IBM i Access for Windows system object API 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.

```

### **Related reference**

“System Objects APIs for IBM i Access for Windows” on page 232

System objects for IBM i Access for Windows application programming interfaces (APIs) allow you to work with print-related objects that are on the system. These APIs make it possible to work with IBM i spooled files, writer jobs, output queues, printers, and more.

*Remote Command/Distributed Program Call APIs return codes:*

There are IBM i Access for Windows Remote command and distributed program call API 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
      Library name is too long or cannot be converted.
6006  CWBRC_INVALID_TYPE
      Invalid parameter type specified.
6007  CWBRC_INVALID_PARM_LENGTH
      Invalid parameter length.

```

6008 CWBRC\_INVALID\_PARM  
Invalid parameter specified.

6009 CWBRC\_TOO\_MANY\_PARAMS  
Attempt to add too many 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 too long.

6099 CWBRC\_UNEXPECTED\_ERROR  
Unexpected error.

### Related reference

“IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338  
The IBM i Access for Windows Remote Command/Distributed Program Call APIs allow the PC application programmer to access IBM i functions. User program and system commands are called without requiring an emulation session. A single IBM i program serves commands and programs, so only one system job is started for both.

*Security APIs return codes:*

There are IBM i Access for Windows security API 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 server user profile is not valid.

6011 CWBSY\_USER\_PROFILE\_DISABLED  
The IBM i 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 IBM i security server. This could indicate tampering with the IBM supplied security server program on the system.

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\_VerifyUserIDPwd 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 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 system value, QPWDMINLEN.

6259 CWBSY\_PWD\_REPEAT\_CHARACTER  
The new password contains a character used more than once. The IBM i 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 IBM i 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 IBM i 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 IBM i 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. IBM i 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 IBM i 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 server configuration (system value QPWDPDIF) 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 server.

6270 CWBSY\_LAST\_INVALID\_PASSWORD  
The next invalid will disable the user profile.

### Related reference

“IBM i Access for Windows Communications and Security APIs” on page 43  
The IBM i Access for Windows Communications and Security topic shows you how to use IBM i Access for Windows application programming interfaces (APIs)

*Serviceability APIs return codes:*

There are IBM i Access for Windows serviceability API 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.  
6006 CWBSV\_LOG\_NOT\_STARTED  
The requested log could not be started.

### Related reference

“IBM i Access for Windows Serviceability APIs” on page 358

The IBM i Access for Windows Serviceability application programming interfaces (APIs) allow you to log service file messages and events within your program.

*System Object Access APIs return codes:*

There are IBM i Access for Windows SOA API 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

24        Sorting is not allowed for this type of list.  
 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 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.

**Related reference**

“IBM i Access for Windows System Object Access (SOA) APIs” on page 413  
 System Object Access enables you to view and manipulate system objects through a graphical user interface.  
 “About System Object Access errors” on page 422  
 IBM i Access for Windows functions support all System Object Access APIs which use return codes to report error conditions.

**IBM i Access for Windows Administration APIs**

These APIs provide functions that access information about the IBM i Access for Windows code that is installed on the PC.

Administration APIs allow you to determine:

- The IBM i Access for Windows version and service level
- The install status of individual features
- The install status of System i<sup>®</sup> Navigator plug-ins

**IBM i Access for Windows Administration APIs required files:**

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

**Programmer’s Toolkit:**

The IBM i Access for Windows 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**.

**IBM i Access for Windows Administration APIs topics:**

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

## Related reference

“Administration APIs return code” on page 21

There is an IBM i Access for Windows administration return code.

“IBM i name formats for connection APIs” on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

“OEM, ANSI, and Unicode considerations” on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

## Administration APIs list

The following APIs are used with IBM i Access for Windows Administration.

### **cwbAD\_GetClientVersion:**

Use the IBM i Access for Windows cwbAD\_GetClientVersion command.

### **Purpose**

Get the version of the IBM i 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 IBM i Access for Windows product is returned.

#### **unsigned long \*release - output**

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

#### **unsigned long \*modificationLevel - output**

Pointer to a buffer where the modification level of the IBM i 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:**

Use the IBM i Access for Windows cwbAD\_GetProductFixLevel command.

## Purpose

Returns the IBM i Access for Windows current fix level.

## 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 IBM i Access for Windows product. Returns an empty string if fixes have not been applied.

### **cwbAD\_IsComponentInstalled:**

IBM i Access for Windows components are called features. Use this API to identify installed features of the product.

## Purpose

Indicates whether a specific IBM i Access for Windows feature 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

**Note:** This constant is defined to be the same as CWBAD\_COMP\_SSL.

**CWB\_COMP\_BASESUPPORT**  
IBM i Access for Windows required programs

**CWBAD\_COMP\_OPTIONAL\_COMPS**  
IBM i Access for Windows optional features

**CWBAD\_COMP\_DIRECTORYUPDATE**  
Directory Update

**CWBAD\_COMP\_IRC**  
Incoming Remote Command

**CWBAD\_COMP\_OUG**  
User's Guide

**CWBAD\_COMP\_OPNAV**  
System i 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\_MP**  
.NET Data 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\_STANDARD
- 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\_PC5250\_FONT\_HINDI

### **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\_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**

System i Navigator Base Support

### **CWBAD\_COMP\_OPNAV\_BASE\_OPS**

System i Navigator Basic Operations

### **CWBAD\_COMP\_OPNAV\_JOB\_MGMT**

System i Navigator Job Management

### **CWBAD\_COMP\_OPNAV\_SYS\_CFG**

System i Navigator System Configuration

### **CWBAD\_COMP\_OPNAV\_NETWORK**

System i Navigator Networks

### **CWBAD\_COMP\_OPNAV\_SECURITY**

System i Navigator Security

### **CWBAD\_COMP\_OPNAV\_USERS\_GROUPS**

System i Navigator Users and Groups

**CWBAD\_COMP\_OPNAV\_DATABASE**

System i Navigator Database

**CWBAD\_COMP\_OPNAV\_BACKUP**

System i Navigator Backup

**CWBAD\_COMP\_OPNAV\_APP\_DEV**

System i Navigator Application Development

**CWBAD\_COMP\_OPNAV\_APP\_ADMIN**

System i Navigator Application Administration

**CWBAD\_COMP\_OPNAV\_FILE\_SYSTEMS**

System i Navigator File Systems

**CWBAD\_COMP\_OPNAV\_MGMT\_CENTRAL**

System i Navigator Management Central

**CWBAD\_COMP\_OPNAV\_MGMT\_COMMANDS**

System i Navigator Management Central - Commands

**CWBAD\_COMP\_OPNAV\_MGMT\_PACK\_PROD**

System i Navigator Management Central - Packages and Products

**CWBAD\_COMP\_OPNAV\_MGMT\_MONITORS**

System i Navigator Management Central - Monitors

**CWBAD\_COMP\_OPNAV\_LOGICAL\_SYS**

System i Navigator Logical Systems

**CWBAD\_COMP\_OPNAV\_ADV\_FUNC\_PRES**

System i 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:**

Use the IBM i Access for Windows cwbAD\_IsOpNavPluginInstalled command.

**Purpose**

Indicates whether a specific System i 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 IBM i Access for Windows Administration APIs.

In this example, the APIs are used to get and display:

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

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

## Usage notes:

Include cwbad.h \*

Link with cwbapi.lib

## Example

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

#include "cwbad.h"

/*
 * This is the highest numbered component ID known (it is
 * the ID of the last component defined in cwbad.h).
 */
#define LAST_COMPID_WE_KNOW_ABOUT      (CWBAD_COMP_SSL)

/*
 * Array of component names, taken from comments for component IDs
 * in cwbad.h, so human-readable component descriptions are displayed .
 * 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, a string is provided 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 Features",
    "Directory Update",
    "Incoming Remote Command",
    "", // not used,
    "Online User's Guide",
    "System i 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",
    "IBM i Java Toolbox",
    "5250 Display and Printer Emulator",
    "Printer Drivers",
    "AFP printer driver",
    "SCS printer driver",
    "IBM i Operations Console",
    "IBM i Access Programmer's Toolkit",
    "Headers, Libraries, and Documentation",
    "", // not used,
    "Java Toolkit",
    "Screen customizer",
    ".NET Data Provider",
    "", //-----#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
    "System i Navigator Base Support",
    "System i Navigator Basic Operations",
    "System i Navigator Job Management",
    "System i Navigator System Configuration",
    "System i Navigator Networks",
    "System i Navigator Security",
    "System i Navigator Users and Groups",
    "System i Navigator Database",
    "", // not used #108
    "System i Navigator Backup",
    "System i Navigator Application Development",
    "System i Navigator Application Administrat",
    "System i Navigator File Systems",
    "System i Navigator Management Central",
    "System i Navigator Management Central - Commands",
    "System i Navigator Management Central - Packages and Products",
    "System i Navigator Management Central - Monitors",
    "System i Navigator Logical Systems",

```

```

"System i Navigator Advanced Function Presentation",
"", //-----#119
"", "", "", "", "", // not #120-124
"", "", "", "", "", // #125-129
"", "", "", "", "", // #130-134
"", "", "", "", "", // used #135-139
"", "", "", "", "", // #140-144
"", "", "", "", "", //----- #145-149
"PC5250: BASE_KOREAN",
"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",
"PC5250: FONT_HINDI",
"", //----- #169
"", "", "", "", "", // #170-174
"", "", "", "", "", // not #175-179
"", "", "", "", "", // #180-184
"", "", "", "", "", // used #185-189
"", "", "", "", "", // #190-194
"", "", "", "", "", //----- #195-199
"Secure Sockets Layer (SSL)" } ; // last one defined
static char unknownComp[] = "unknown, ID=" ;
static char* pInsertID = &( unknownComp[12] ); // insert ID here!

```

```

/*****
 * Show the IBM i 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( "IBM i 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( "IBM i 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 IBM i 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 known.
 *****/

```

```

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 its name is not known,
     *         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, and a name is known, return it
     */
    else
    {
        pCompName = compDescr[ compID ];
    }

    return pCompName;
}

```

```

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

    printf( "IBM i Access features installed:\n\n" );

    /*
     * Try all known features, plus a bunch more in case some
     * have been added (via service pack).
     */
    for ( compID = 0;
          compID < (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( "IBM i 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, 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" );
}

```

## IBM i Access for Windows Communications and Security APIs

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

You can use these APIs to:

- Get, use, and delete an IBM i **system object**. Various IBM i Access for Windows APIs require a system object. It holds information about connecting to, and validating IBM i security objects including user ID, password, and signon date and time .
- Obtain information about environments and connections that are configured in the **system list** when you use IBM i Access for Windows functions. 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.

### IBM i Access for Windows Communications and Security APIs required files:

Header file		Import library	Dynamic Link Library
System object APIs	System list APIs	cwbapi.lib	cwbcodll
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**.

### IBM i Access for Windows Communications and Security topics:

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

## Related reference

“Communications APIs return codes” on page 21

There are IBM i Access for Windows communications API return codes.

“Security APIs return codes” on page 29

There are IBM i Access for Windows security API return codes.

“Global IBM i Access for Windows return codes” on page 14

There are global IBM i Access for Windows return codes.

“IBM i name formats for connection APIs” on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

“OEM, ANSI, and Unicode considerations” on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

## System object attributes

System object attributes, on the IBM i platform, affect the behavior of signing on and communicating with the system represented by the system object.

Most attributes are not changeable after a successful signon has occurred using either `cwbCO_Signon` or `cwbCO_Connect`. The only two attributes that are changeable after a successful signon are the Window Handle and the Connect Timeout attributes. Calling an API to change the value of other attributes, after a successful signon, fails with return code `CWB_INV_AFTER_SIGNON`.

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 list 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 system that is named 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.

## Related reference

“`cwbCO_Signon`” on page 76

Use the IBM i Access for Windows `cwbCO_Signon` command.

“`cwbCO_Connect`” on page 50

Use the IBM i Access for Windows `cwbCO_Connect` command.

## System object attributes list:

Following is a list of IBM i descriptions, requirements, and considerations of system object attributes.

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 system name.

## IBM i name:

The system with which to communicate, as defined by this instance of the 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 unit, 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 IBM i unit. 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 IBM i connection.

Set using System i Navigator.

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 System i 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 IBM i user ID that is used the system.

Get by using `cwbCO_GetUserIDEx`

Set by using `cwbCO_SetUserIDEx`

**Default:**

The first time that you connect to the system which is named in the system object, you might 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 IBM i password used to signon to the 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 system that is named in the system object. If a previous successful signon or connection has been made to the 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 IBM i 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 IBM i Access for Windows prompting 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 IBM i Access for Windows prompting that does relate to IBM i 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 IBM i communication 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 IBM i Access for Windows sockets are used to authenticate the 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 in the IBM i the System List will be used for this system. If no IBM i configuration for this system exists, or if the configuration specifies to use the IBM i 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 IBM i host service. It specifies whether to look it up locally (on the PC), on the IBM i host, 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 IBM i mapper is made to retrieve the port number by lookup from the IBM i 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 system in the IBM i List is used. If no IBM i configuration exists for this system, the default is `CWBCO_PORT_LOOKUP_SERVER`.

**Note:** The default may be overridden by policies.

**Persistence mode:**

Specifies whether the system named in this system object may be added to the IBM i 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 IBM i Access for Windows wait time for the completion of a connection attempt. 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 IBM i 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.

**Communications and security: Create and delete APIs**

These APIs are used for creating and deleting an IBM i object

**`cwbCO_CreateSystem:`**

Use the IBM i Access for Windows `cwbCO_CreateSystem` command.

## 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 44 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 IBM i name. This can be its host name, or the IBM i dotted-decimal IP address itself. It must not be zero length and must not contain blanks. If the name specified is not a valid IBM i 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`.

## Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

## cwbCO\_CreateSystemLike:

Use the IBM i Access for Windows `cwbCO_CreateSystemLike` command.

## 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 list" on page 44 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 is the IBM i identification. This is the object that will be "copied."

### **LPCSTR systemName - input**

Pointer to a buffer that contains the NULL-terminated IBM i name 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 IBM i dotted-decimal IP address. If the name that is specified is not a valid IBM i 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:**

Use the IBM i Access for Windows `cwbCO_DeleteSystem` command.

#### **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 is the IBM i identification.

#### **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.

### **Communications and security: Connect and disconnect APIs**

These APIs support IBM i connection and disconnection, and other related behaviors.

#### **cwbCO\_Connect:**

Use the IBM i Access for Windows `cwbCO_Connect` command.

#### **Purpose**

Connect to the specified IBM i 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 is the IBM i identification that will be used for the connection.

### **cwbCO\_Service service - input**

The IBM i service for the connection. Valid values are those listed in “Defines for `cwbCO_Service`” on page 95, 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 system, so the attempt timed out.

### **CWB\_CONNECTION\_REFUSED**

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

### **CWB\_USER\_TIMEOUT**

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

### **CWB\_FIPS\_UNAVAILABLE**

This connection is configured for SSL and FIPS-compliant mode is enabled, however, SSL cannot be used because FIPS support is not available. For recovery information, see message `CWBCO1060`, using the following path:

**Start → Programs → IBM i Access for Windows → User’s Guide → Messages → IBM i Access for Windows messages → CWBCO1060**

**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 the IBM i signon 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 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 system, as named in the system object, is 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 made through the specified system object, and reset all usage counts to 0.

If the return code is `CWB_USER_TIMEOUT`, you may want to increase the connect timeout value for this system object, by calling `cwbCO_SetConnectTimeout`, and try connecting again. If you want IBM i 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.

## Related reference

“System object attributes” on page 44

System object attributes, on the IBM i platform, affect the behavior of signing on and communicating with the system represented by the system object.

## `cwbCO_Disconnect`:

Use the IBM i Access for Windows `cwbCO_Disconnect` command.

## Purpose

Disconnect from the specified IBM i host service.

## Syntax

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

## Parameters

### `cwbCO_SysHandle` system - input

Handle that was returned by a previous call to either `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It the IBM i identification used for the disconnect.

### **cwbCO\_Service service - input**

The IBM i service for disconnect. 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 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\_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.

IBM i disconnect message might be generated when requesting that all active services be disconnected.

#### **cwbCO\_GetConnectTimeout:**

Use the IBM i Access for Windows cwbCO\_GetConnectTimeout command.

### **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 is the IBM i identification.

### **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\_GetPersistenceMode:**

Use the IBM i Access for Windows `cwbCO_GetPersistenceMode` command.

## 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 is the IBM i identification.

### **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\_IsConnected:**

Use the IBM i `cwbCO_IsConnected` command.

## Purpose

Find out if any, and how many, IBM i connections 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 is the IBM i identification.

### **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 95 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\_SetConnectTimeout:**

Use the IBM i Access for Windows cwbCO\_SetConnectTimeout command.

### **Purpose**

This function sets, for the specified system object, the IBM i Access for Windows wait time, in seconds that the product waits 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 is the IBM i identification.

#### **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\_SetPersistenceMode:**

Use the IBM i Access for Windows `cwbCO_SetPersistenceMode` command.

## 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 is the IBM i identification.

### **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_Verify`:

Use the IBM i Access for Windows `cwbCO_Verify` command.

## Purpose

Verifies that a connection can be made to a specific IBM i host service.

## Syntax

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

## Parameters

### `cwbCO_SysHandle system` - input

Handle previously returned from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It is the IBM i identification whose connectability is verified.

### `cwbCO_Service service` - input

The IBM i service whose connectability is verified. Valid values are those listed in “Defines for `cwbCO_Service`” on page 95, except for the value `CWBCO_SERVICE_ANY`. To verify connectability of ALL services, specify `CWBCO_SERVICE_ALL`.

### `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 `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\_USER\_TIMEOUT**

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

### **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.

## Communication and security: Security validation and data APIs

These IBM i APIs provide security validation and data.

### **cwbcO\_ChangePassword:**

Use the IBM i Access for Windows cwbcO\_ChangePassword command.

### **Purpose**

Changes the password of the specified IBM i user 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,  
    cwbcSV_ErrHandle  errorHandle);
```

### **Parameters**

#### **cwbcO\_SysHandle system - input**

Handle returned previously from cwbcO\_CreateSystem or cwbcO\_CreateSystemLike. This is the IBM i identification.

#### **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.

#### **cwbcSV\_ErrHandle errorHandle - input/output**

Any returned messages will be written to this object. It is created with the cwbcSV\_CreateErrHandle API. The messages may be retrieved through the cwbcSV\_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 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 IBM i 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\_GetDefaultUserMode:**

Use the IBM i Access for Windows cwbCO\_GetDefaultUserMode command.

**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. This is the IBM i identification.

**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\_GetFailedSignons:**

Use the IBM i Access for Windows `cwbCO_GetFailedSignons` command.

### 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 is the IBM i identification.

#### **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\_GetPasswordExpireDate:**

Use the IBM i `cwbCO_GetPasswordExpireDate` command.

### Purpose

Retrieves the date and time the password expires for the IBM i user ID, for the system that is specified by the system object.

### 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 is the IBM i identification.

### **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 (so the password expire date is not available), or validation has occurred and the user profile password expiration interval is set to \*NOMAX.

## 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`.

If the user profile password expiration interval is set to \*NOMAX, a password expire date does not exist. To detect this case, first validate the user ID and password as noted above, and then, if successful, call `cwbCO_GetPasswordExpireDate`. A return code of `CWBCO_INV_BEFORE_VALIDATE` means that the password expiration interval is set to \*NOMAX.

### **cwbCO\_GetPrevSignonDate:**

Use the IBM i Access for Windows `cwbCO_GetPrevSignonDate` command.

## 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 is the IBM i identification.

### **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:**

Use the IBM i Access for Windows cwbCO\_GetPromptMode command.

### **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\_CreateSystemLikeIt is the IBM i identification.

#### **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:**

Use the IBM i Access for Windows cwbCO\_GetSignonDate command.

### **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 is the IBM i identification.

### **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\_GetUserIDEx:**

Use the IBM i Access for Windows cwbCO\_GetUserIDEx command.

### **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 IBM i connection.

### **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 is the IBM i identification.

### **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 IBM i user ID might or might not have been validated 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 an IBM i default user ID is configured.

### **cwbCO\_GetValidateMode:**

Use the IBM i Access for Windows `cwbCO_GetValidateMode` command.

## 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 is the IBM i identification.

### **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:**

Use the IBM i Access for Windows cwbCO\_GetWindowHandle command.

## 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 is the IBM i identification.

### **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:**

Use the IBM i Access for Windows `cwbCO_HasSignedOn` command.

### 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 is the IBM i identification.

#### **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 might not have caused or included an IBM i connection and security validation flow. 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, might 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` is set to `CWB_FALSE`.

#### **cwbCO\_SetDefaultUserMode:**

Use the IBM i Access for Windows `cwbCO_SetDefaultUserMode` command.

### 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 is the IBM i identification.

### **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 System 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 IBM i default user ID that is configured for the 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 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\_SetPassword:**

Use the IBM i Access for Windows `cwbCO_SetPassword` command.

### **Purpose**

This function sets the password to associate with the specified system object. This password is used for an IBM i connection with either the `cwbCO_Signon` or `cwbCO_Connect` call, and when a user ID is 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 is the IBM i identification.

#### **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 IBM i 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\_SetPromptMode:**

Use the IBM i Access for Windows `cwbCO_SetPromptMode` command.

## 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 is the IBM i identification.

### **cwbCO\_PromptMode - input**

Specifies the prompt mode. Possible values are:

#### **CWBCO\_PROMPT\_IF\_NECESSARY**

IBM i Access for Windows prompting occurs if either the user ID or password are not explicitly set or cannot be retrieved from the persistent configuration for this system, from the password cache (if enabled), or by some other means.

If the Default User Mode is set, and if IBM i prompting has not occurred for the default user ID, IBM i prompting occurs for it at `cwbCO_Connect` or `cwbCO_Signon` time

#### **CWBCO\_PROMPT\_ALWAYS**

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

#### **CWBCO\_PROMPT\_NEVER**

IBM i Access for Windows prompting never occurs for the user ID and password, or for the 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 are not set and cannot be programmatically retrieved (from the IBM i password cache). This mode is used when either

- The IBM i Access for Windows product 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:**

Use the IBM i Access for Windows `cwbCO_SetUserIDEx` command.

## Purpose

This function sets the user ID to associate with the specified system object. This user ID is used on the IBM i connection 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 is the IBM i identification.

### **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\_SetWindowHandle:**

Use the IBM i Access for Windows `cwbCO_SetWindowHandle` command.

## 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 is the IBM i identification.

### **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\_SetValidateMode:**

Use the IBM i Access for Windows cwbCO\_SetValidateMode command.

### **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 is the IBM i identification.

#### **cwbCO\_ValidateMode mode - input**

Specifies the validate mode. Possible values are:

##### **CWBCO\_VALIDATE\_IF\_NECESSARY**

If validation of this IBM i user ID 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 might be other scenarios where re-validation occurs. IBM i Access for Windows re-validation occurs as needed.

##### **CWBCO\_VALIDATE\_ALWAYS**

IBM i communication to validate user ID and password occurs 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\_Signon:

Use the IBM i Access for Windows `cwbCO_Signon` command.

### Purpose

Use the user ID and password to sign on the user to the system that is represented by the IBM i specified object.

**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 is the IBM i identification.

#### **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 IBM i prompting for user password and actual IBM i contact 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 IBM i name 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 IBM i name that is in the system object, is added to the System List.

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

See also:

- "Differences between `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword`" on page 96
- "Similarities between `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword`" on page 96

**Related reference**

"System object attributes" on page 44

System object attributes, on the IBM i platform, affect the behavior of signing on and communicating with the system represented by the system object.

**`cwbCO_VerifyUserIDPassword`:**

Use the IBM i Access for Windows `cwbCO_VerifyUserIDPassword` command.

**Purpose**

This function verifies the correctness of the IBM i user ID and password, on the system represented by the specified system object. 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,  
    cwbcSV_ErrHandle  errorHandle );
```

## Parameters

### **cwbcO\_SysHandle system - input**

Handle that previously was returned from `cwbcO_CreateSystem` or `cwbcO_CreateSystemLike`. It is the IBM i identification.

### **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.

### **cwbcSV\_ErrHandle errorHandle - input/output**

Any returned messages will be written to this object. It is created with the `cwbcSV_CreateErrHandle` API. The messages may be retrieved through the `cwbcSV_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 IBM i 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 96 and “Similarities between `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword`” on page 96.

## Communications and security: Get and set attribute APIs

Use the IBM i Access for Windows APIs to get and set other system object attributes, or determine if the attributes are restricted by policies.

### `cwbCO_CanModifyDefaultUserMode`:

Use the IBM i Access for Windows `cwbCO_CanModifyDefaultUserMode` command.

## 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 is the IBM i identification.

### `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:**

Use the IBM i Access for Windows cwbCO\_CanModifyIPAddress command.

### **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 is the IBM i identification.

#### **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:**

Use the IBM i Access for Windows cwbCO\_CanModifyIPAddressLookupMode command.

### **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 is the IBM i identification.

### **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:**

Use the IBM i Access for Windows `cwbCO_CanModifyPersistenceMode` command.

## 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 is the IBM i identification.

### **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:**

Use the IBM i Access for Windows cwbCO\_CanModifyPortLookupMode command.

## 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 is the IBM i identification.

### **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:**

Use the IBM i Access for Windows `cwbCO_CanModifyUseSecureSockets` command.

## 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 is the IBM i identification.

### **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\_GetDescription:**

Use the IBM i Access for Windows `cwbCO_GetDescription` command.

## Purpose

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

## Syntax

```
UINT CWB_ENTRY cwbCO_GetDescription(  
                                cwbCO_SysHandle  system,  
                                LPSTR            description,  
                                PULONG           length );
```

## Parameters

### **cwbCO\_SysHandle system - input**

Handle returned previously from `cwbCO_CreateSystem` or `cwbCO_CreateSystemLike`. It is the IBM i identification.

### **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\_GetHostCCSID:**

Use the IBM i Access for Windows `cwbCO_GetHostCCSID` command.

## Purpose

Returns the IBM i associated CCSID that is represented by the user ID that is in the system object, that was in use when the signon to the system occurred.

## 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 is the IBM i identification.

### **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.

### **CWB\_USER\_TIMEOUT**

### **CWB\_SSL\_JAVA\_ERROR**

### **CWB\_USER\_TIMEOUT\_SENDRCV**

## 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 IBM i default CCSID. To retrieve a host CCSID without requiring a user ID, call `cwbNL_GetHostCCSID`.

### **cwbCO\_GetHostVersionEx:**

Use the IBM i Access for Windows `cwbCO_GetHostVersionEx` command.

## 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 is the IBM i identification.

**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 an IBM i connection is made. If an IBM i connection does not exist in the currently-active environment, this information is not available, and the error code `CWB_NOT_CONNECTED` is returned. If you know that a successful IBM i connection was made, it is likely that the version and release levels returned are current. If you want to make sure that the values are available and have been recently retrieved, call `cwbCO_Signon` or `cwbCO_Connect` for this system object first, then call `cwbCO_GetHostVersionEx`.

**cwbCO\_GetIPAddress:**

Use the IBM i Access for Windows `cwbCO_GetIPAddress` command.

**Purpose**

This function gets the IBM i IP address represented by the specified system object. This is the IP address that was used on the IBM i connection (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 is the IBM i identification.

**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:**

Use the IBM i Access for Windows cwbCO\_GetIPAddressLookupMode command.

### **Purpose**

This function gets the indication of when, if ever, dynamic lookup occurs for the IBM i IP address represented by the specified system object.

### **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 is the IBM i identification.

#### **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\_GetPortLookupMode:**

Use the IBM i Access for Windows cwbCO\_GetPortLookupMode command.

### **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 to establish an IBM i Access for Windows 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 is the IBM i identification.

#### **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\_GetSystemName:**

Use the IBM i Access for Windows cwbCO\_GetSystemName command.

### **Purpose**

This function gets the IBM i 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 is the IBM i identification.

### **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\_IsSecureSockets:**

Use the IBM i Access for Windows `cwbCO_IsSecureSockets` command.

## 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 is the IBM i identification..

### **cwb\_Boolean \* inUse - output**

Returns whether IBM i 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 which IBM i Access for Windows attempts are tried for future communications. If `CWB_TRUE` is returned, then any IBM i attempt to communicate that cannot be performed using secure sockets will fail.

Although with limitations, the IBM i Access for Windows product enforces Federal Information Processing Standards (FIPS) compliance when SSL is used, this API does not return an indication of whether FIPS compliance is on or off. The only way to verify that FIPS-compliance is on or off is to visually inspect the FIPS compliance checkbox in IBM i Access for Windows Properties. For more information about FIPS and its use, see the IBM i Access for Windows User's Guide that is installed with the product.

### **cwbCO\_SetIPAddress:**

Use the IBM i Access for Windows `cwbCO_SetIPAddress` command.

## Purpose

This function sets, for the specified system object, the IP address that will be used for the IBM i connection. 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 is the IBM i identification.

### **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:**

Use the IBM i Access for Windows `cwbCO_SetIPAddressLookupMode` command.

## Purpose

This function sets, for the specified system object, when dynamic lookup occurs for the IBM i IP address when a connection is to be made for the system represented by the specified system object. 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 the IBM i Access for Windows product 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 is the IBM i identification.

### **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 IBM i 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 system.

#### **CWBCO\_IPADDR\_LOOKUP\_1DAY**

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

#### **CWBCO\_IPADDR\_LOOKUP\_1WEEK**

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

#### **CWBCO\_IPADDR\_LOOKUP\_NEVER**

Never dynamically lookup the IBM i IP address of this system. Always use the IP address that was last used on this PC for the system.

#### **CWBCO\_IPADDR\_LOOKUP\_AFTER\_STARTUP**

Lookup the IP address dynamically if Windows has been re-started since the last lookup for this 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 IBM i connection time, since the dynamic lookup might cause network traffic and take many seconds to complete. If the dynamic lookup is not performed, there is a risk that the IBM i IP address has changed and a connection either fails or a connection is made to the wrong system.

## **cwbCO\_SetPortLookupMode:**

Use the IBM i Access for Windows cwbCO\_SetPortLookupMode command.

### **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 is the IBM i identification.

#### **cwbCO\_PortLookupMode mode - input**

Specifies port lookup method. Possible values are:

##### **CWBCO\_PORT\_LOOKUP\_SERVER**

Lookup of a host server port is done by contacting the IBM i host 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 IBM i service.

##### **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 is used to connect to the desired service. The standard port is the port that is set by default for a given host server and is used, if there are not any changes made to the IBM i services table for that service.

The latter two modes eliminate the IBM i mapper connection and its associated delay, network traffic, and load on the 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 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 IBM i host service in the service table on that system, this mode will not work.

Use `CWBCO_PORT_LOOKUP_LOCAL` for best performance when the port for an IBM i Access host service has been changed on the 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 is located in a subdirectory under the Windows install directory called `system32\drivers\etc`.

### **cwbCO\_UseSecureSockets:**

Use the IBM i Access for Windows `cwbCO_UseSecureSockets` command.

## Purpose

Specifies that all IBM i communication to the system represented by the system object must either 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 IBM i system.

### **cwb\_Boolean useSecureSockets - input**

Specifies whether to require secure sockets use when communicating with the 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\_USER\_TIMEOUT**

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 IBM i 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 system, while the secure-sockets version of the service might not be running, or vice-versa.

At the time this API is called, the IBM i Access for Windows product might not detect that Secure Sockets is available for use at IBM i connection time. Even if `CWB_SECURE_SOCKETS_NOTAVAIL` is NOT returned, it might be determined at a later time that secure sockets is not available.

Although with limitations, the IBM i Access for Windows product enforces Federal Information Processing Standards (FIPS) compliance when SSL is used, this API does not return an indication of whether FIPS compliance is on or off. The only way to verify that FIPS-compliance is on or off is to visually inspect the FIPS compliance checkbox in IBM i Access for Windows Properties. For more information about FIPS and its use, see the IBM i Access for Windows User's Guide that is installed with the product.

**Defines for cwbcO\_Service**

The following values define IBM i Access for Windows `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_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 the IBM i Access for Windows `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword` commands..

- `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 an IBM i connection 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 is cached in the IBM i 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`

The following information illustrates the similarities between IBM i Access for Windows `cwbCO_Signon` and `cwbCO_VerifyUserIDPassword` commands.

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`

## Communications: Create and delete APIs

Use these IBM i Access for Windows APIs to 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.

### `cwbCO_CreateSysListHandle`:

Use the IBM i Access for Windows `cwbCO_CreateSysListHandle` command.

#### 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:**

Use the IBM i Access for Windows cwbCO\_CreateSysListHandleEnv command.

## 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 ("\"), 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.

**CWBCO\_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`:**

Use the IBM i Access for Windows `cwbCO_DeleteSysListHandle` command.

**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_GetNextSysName`:**

Use the IBM i Access for Windows `cwbCO_GetNextSysName` command.

## 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\_GetSysListSize:**

Use the IBM i Access for Windows cwbCO\_GetSysListSize command.

### **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.

## **Communications: System information APIs**

Use these IBM i Access for Windows APIs to 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.

## **cwbCO\_GetActiveConversations:**

Use the IBM i Access for Windows cwbCO\_GetActiveConversations command.

### **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 that are active for the specified system within the CURRENT PROCESS ONLY. There may be other conversations active within other processes running on the PC.

### cwbCO\_GetConnectedSysName:

Use the IBM i Access for Windows cwbCO\_GetConnectedSysName command.

## 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:**

Use the IBM i Access for Windows cwbCO\_GetDefaultSysName command.

### **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 errorHandler);
```

### **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 errorHandler - 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**

None.

**cwbCO\_IsSystemConfigured:**

Use the IBM i Access for Windows cwbCO\_IsSystemConfigured command.

**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:**

Use the IBM i Access for Windows cwbCO\_IsSystemConfiguredEnv command.

**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:

Use the IBM i Access for Windows cwbCO\_IsSystemConnected command.

## 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.

## Communications: Configured environments information

Use these IBM i Access for Windows APIs to obtain the names of environments that have been configured.

### **cwbCO\_GetActiveEnvironment:**

Use the IBM i Access for Windows `cwbCO_GetActiveEnvironment` command.

## 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

None.

## cwbCO\_GetEnvironmentName:

Use the IBM i Access for Windows cwbCO\_GetEnvironmentName command.

## 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

None.

### **cwbCO\_GetNumberOfEnvironments:**

Use the IBM i Access for Windows cwbCO\_GetNumberOfEnvironments command.

## Purpose

Get the number of IBM i 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.

## **Communications: Environment and connection information**

Use these IBM i Access for Windows APIs to determine if the calling application can modify environments and connection information.

### **cwbCO\_CanConnectNewSystem:**

Use the IBM i Access for Windows cwbCO\_CanConnectNewSystemcommand.

## 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 IBM i Access for Windows APIs that take system name as a parameter.

### **cwbCO\_CanModifyEnvironmentList:**

Use the IBM i Access for Windows `cwbCO_CanModifyEnvironmentList` command.

## 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:**

Use the IBM i Access for Windows `cwbCO_CanModifySystemList` command.

## 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:**

Use the IBM i Access for Windows `cwbCO_CanModifySystemListEnv` command.

## 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:**

Use the IBM i Access for Windows `cwbCO_CanSetActiveEnvironment` command.

## 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

## Example: Using IBM i Access for Windows communications APIs

The example program below shows the use of IBM i Access for Windows 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

```

-----
5770-XE1 (C) Copyright IBM CORP. 1996, 2009
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 "cwbc.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* df1tUserModeStr[5] = { "CWBCO_DEFAULT_USER_MODE_NOT_SET" ,
                              "CWBCO_DEFAULT_USER_USE" ,
                              "CWBCO_DEFAULT_USER_IGNORE" ,
                              "CWBCO_DEFAULT_USER_USEWINLOGON" ,
                              "CWBCO_DEFAULT_USER_USE_KERBEROS" } ;

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" ,
                        "", /* not used */
                        "", /* not used */
                        "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 )
{
    PSYSLISTNODE 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=0, 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", numSystems );
            *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,
    cwb_Boolean* pCanBeModified,
    UINT canBeModifiedRC )
{
    if ( CWB_OK == lookupRC )
    {
        printf( "%25s : %-30s  ", label, attrVal );
        if ( CWB_OK == canBeModifiedRC )
        {
            if ( pCanBeModified != NULL )
            {
                printf( "%s\n", cwbBoolStr[ *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", cwBoolStr[ 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, &portLookupMode );
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" );
}

```

## IBM i Data Queues APIs

Use IBM i Access for Windows Data Queues application programming interfaces (APIs) to provide easy access to IBM i data queues. Data queues allow you to create client/server applications that do not require the use of communications APIs.

### IBM i 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**.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

## Related reference

“Data Queues APIs return codes” on page 25

There are IBM i Access for Windows data queues API return codes.

“IBM i name formats for connection APIs” on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

“OEM, ANSI, and Unicode considerations” on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

## Data queues

A data queue is an IBM i object.

### Benefits of using data queues:

Data queues provide many benefits to PC developers and IBM i applications developers, including:

- They are a fast and efficient means of IBM i communication.
- 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 IBM i 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 an IBM i 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.

## Work with data queues

You can work with data queues by using IBM i CL commands or callable programming interfaces. Access to data queues is available to all IBM i applications regardless of the programming language in which the application is written.

Use the following IBM i interfaces to work with data queues:

### IBM i commands:

#### CRTDTAQ

Creates a data queue and stores it in a specified library

#### DLTDTAQ

Deletes the specified data queue from the system

### IBM i 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 IBM i programming 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 IBM i data queue.
- A partner program (either a PC program or an IBM i 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 IBM i 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 IBM i data queue.

### Attribute:

This object describes the IBM i data queue.

**Data:** Use these objects to write records to, and to read records from, the IBM i data queue.

### Read object:

Use this object only with the asynchronous read APIs. It uniquely identifies a request to read a record from the IBM i data queue. This handle is used on subsequent calls to check if the data has been returned. See `thecwbDQ_AsyncRead` API for more information.

## Related reference

“cwbDQ\_AsyncRead” on page 129

Use the IBM i Access for Windows cwbDQ\_AsyncRead command.

## Data Queues: Create, delete, and open APIs

Use these IBM i APIs in conjunction with the cwbCO\_SysHandle System Object handle.

### cwbDQ\_CreateEx:

Use the IBM i Access for Windows cwbDQ\_CreateEx command.

### Purpose

Create an IBM i 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 errorHandle);
```

### Parameters

#### **cwbCO\_SysHandle sysHandle - input**

Handle to a system object

#### **const char \* queue - input**

Pointer to the data queue name contained in an ASCII string.

#### **const char \* library - input**

Pointer to the library name contained in an ASCII 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 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**

IBM i application not found.

#### **CWB\_HOST\_NOT\_FOUND**

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\_DeleteEx:**

Use the IBM i Access for Windows cwbDQ\_DeleteEx command.

### **Purpose**

Remove all data from an IBM i 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**

IBM i application not found.

#### **CWB\_HOST\_NOT\_FOUND**

System is 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_OpenEx`:**

Use the IBM i Access for Windows `cwbDQ_OpenEx` command.

**Purpose**

Start a connection to the specified data queue. This will start an IBM i conversation. 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**

IBM i application is not found.

**CWB\_HOST\_NOT\_FOUND**

System is 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`.

**Data Queues: Accessing data queues APIs**

After the `cwbDQ_Open` API is used to create a connection to a specific IBM i data queue, these other APIs are called to utilize it. Use the `cwbDQ_Close` API when the connection no longer is needed.

**`cwbDQ_AsyncRead`:**

Use the IBM i Access for Windows `cwbDQ_AsyncRead` command.

**Purpose**

Read a record from the IBM i 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 IBM i data queue object.

**cwbDQ\_Data data - input**

The data object to be read from the IBM i 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

**Related concepts**

“Typical use of data queues” on page 123

A data queue is a powerful program-to-program interface. Programmers who are familiar with IBM i programming are accustomed to using queues. Data queues simply represent a method that is used to pass information to another program.

**cwbDQ\_Cancel:**

Use the IBM i Access for Windows cwbDQ\_Cancel command.

**Purpose**

Cancel a previously issued AsyncRead. This will end the read on the IBM i data queue.

**Syntax**

```
unsigned int CWB_ENTRY cwbDQ_Cancel(
    cwbDQ_ReadHandle readHandle,
    cwbSV_ErrHandle errorHandler);
```

## 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.

### **CWB\_DQ\_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:**

Use the IBM i Access for Windows cwbDQ\_CheckData command.

## 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`:**

Use the IBM i Access for Windows `cwbDQ_Clear` command.

**Purpose**

Remove all messages from the IBM i 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    errorHandle);

```

**Parameters****`cwbDQ_QueueHandle queueHandle` - input**

Handle that was returned by a previous call to the `cwbDQ_Open` function. This identifies the IBM i data queue object.

**`unsigned char * key` - input**

Pointer to the key. The key may contain embedded `NULL`s, 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:**

Use the IBM i Access for Windows `cwbDQ_Close` command.

**Purpose**

End the connection with the IBM i data queue object that is identified by the specified handle. This will end the IBM i conversation.

**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 IBM i data queue object.

**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 previously issued the following APIs:

- `cwbdQ_Open` or `cwbdQ_OpenEx`

### **`cwbdQ_GetLibName`:**

Use the IBM i Access for Windows `cwbdQ_GetLibName` command.

## 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 IBM i 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.

### **`CWBBDQ_INVALID_QUEUE_HANDLE`**

Invalid queue handle.

## Usage

This function requires that you have previously issued `cwbdQ_Open`.

### **`cwbdQ_GetQueueAttr`:**

Use the IBM i Access for Windows `cwbdQ_GetQueueAttr` command.

## Purpose

Retrieve the attributes of the IBM i 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,  
    cwbdQ_ErrHandle errorHandler);
```



## Parameters

### **cwbDQ\_QueueHandle queueHandle - input**

Handle that was returned by a previous call to the `cwbDQ_Open` function. This identifies the IBM i 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 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.

### **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:**

Use the IBM i Access for Windows `cwbDQ_GetQueueName` command.

## 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 IBM i 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\_GetSysName:**

Use the IBM i Access for Windows `cwbdQ_GetSysName` command.

## 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 IBM i 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.

### **CWBDQ\_INVALID\_QUEUE\_HANDLE**

Invalid queue handle.

## Usage

This function requires that you previously have issued `cwbdQ_Open` or `cwbdQ_OpenEx`.

### **cwbdQ\_Peek:**

Use the IBM i Access for Windows `cwbdQ_Peek` command.

## Purpose

Read a record from the IBM i 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,  
    cwbSV_ErrHandle errorHandle);
```

## Parameters

### **cwbDQ\_QueueHandle queueHandle - input**

Handle that was returned by a previous call to the `cwbDQ_Open` API. This identifies the IBM i data queue object.

### **cwbDQ\_Data data - input**

The data object to be read from the IBM i 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.

### **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\_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:**

Use the IBM i Access for Windows `cwbDQ_Read` command.

## Purpose

Read a record from the IBM i 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 IBM i data queue object.

### **cwbDQ\_Data data - input**

The data object to be read from the IBM i 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\_Write:**

Use the IBM i Access for Windows `cwbdQ_Write` command.

**Purpose**

Write a record to the IBM i data queue object that is identified by the specified handle.

**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 IBM i data queue object.

**cwbdQ\_Data data - input**

The data object to be written to the IBM i data queue.

**cwb\_Boolean commit - input**

This flag is no longer used and is ignored.

**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`.

### **Data Queues: Attributes APIs**

Use these APIs to declare attributes of an IBM i data queue. The attribute object is used when creating a data queue or when obtaining the data queue attributes.

#### **`cwbdQ_CreateAttr`:**

Use the IBM i Access for Windows `cwbdQ_CreateAttr` command.

### **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_DeleteAttr`:**

Use the IBM i Access for Windows `cwbDQ_DeleteAttr` command.

### 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.

#### **CWBDQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

### Usage

None

#### **cwbDQ\_GetAuthority:**

Use the IBM i Access for Windows `cwbDQ_GetAuthority` command.

### 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.

### **CWBBDQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

## Usage

None

### **cwbDQ\_GetDesc:**

Use the IBM i Access for Windows cwbDQ\_GetDesc command.

## 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.

### **CWBBDQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

## Usage

None

### **cwbDQ\_GetForceToStorage:**

Use the IBM i Access for Windows cwbDQ\_GetForceToStorage command.



## 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.

### **CWB\_DQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

## Usage

None

### **cwbDQ\_GetKeySize:**

Use the IBM i Access for Windows `cwbDQ_GetKeySize` command.

## 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.

**CWBDQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

**Usage**

None

**cwbDQ\_GetMaxRecLen:**

Use the IBM i Access for Windows cwbDQ\_GetMaxRecLen command.

**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.

**CWBDQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

**Usage**

None

**cwbDQ\_GetOrder:**

Use the IBM i Access for Windows cwbDQ\_GetOrder command.

**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\_GetSenderID:**

Use the IBM i Access for Windows cwbDQ\_GetSenderID command.

### 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.

**CWBDQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

**Usage**

None

**cwbDQ\_SetAuthority:**

Use the IBM i Access for Windows cwbDQ\_SetAuthority command.

**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 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\_SetDesc:**

Use the IBM i Access for Windows cwbDQ\_SetDesc command.

### **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 ASCII 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:**

Use the IBM i Access for Windows cwbDQ\_SetForceToStorage command.

### **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.

### **CWBDQ\_INVALID\_ATTRIBUTE\_HANDLE**

Invalid attributes handle.

## Usage

None

### **cwbDQ\_SetKeySize:**

Use the IBM i Access for Windows `cwbDQ_SetKeySize` command.

## 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:**

Use the IBM i Access for Windows cwbDQ\_SetMaxRecLen command.

### **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:**

Use the IBM i Access for Windows cwbDQ\_SetOrder command.

### **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\_SetSenderID:**

Use the IBM i Access for Windows `cwbDQ_SetSenderID` command.

## 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.

## CWBDDQ\_INVALID\_ATTRIBUTE\_HANDLE

Invalid attributes handle.

## Usage

None

## Data Queues: Read and write APIs

Use these IBM i Access for Windows APIs for writing to and reading from a data queue

### **cwbDQ\_CreateData:**

Use the IBM i Access for Windows cwbDQ\_CreateData command.

## 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\_DeleteData:**

Use the IBM i Access for Windows cwbDQ\_DeleteData command.

## 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.

### **CWBDQ\_INVALID\_DATA\_HANDLE**

Invalid data handle.

## Usage

None

### **cwbdQ\_GetConvert:**

Use the IBM i Access for Windows `cwbdQ_GetConvert` command.

## 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,  
                                cw_Boolean    *convert);
```

## Parameters

### **cwbdQ\_Data data - input**

Handle of the data object that was returned by a previous call to `cwbdQ_CreateData`.

### **cw\_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.

### **CWBDQ\_INVALID\_DATA\_HANDLE**

Invalid data handle.

## Usage

None

### **cwbdQ\_GetData:**

Use the IBM i Access for Windows `cwbDQ_GetData` command.

### 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 ASCII string.

### 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\_GetDataAddr:**

Use the IBM i Access for Windows `cwbDQ_GetDataAddr` command.

### 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:**

Use the IBM i Access for Windows `cwbDQ_GetDataLen` command.

## 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\_GetKey:**

Use the IBM i Access for Windows cwbDQ\_GetKey command.

## 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.

### **CWBBDQ\_INVALID\_DATA\_HANDLE**

Invalid data handle.

## Usage

None

### **cwbDQ\_GetKeyLen:**

Use the IBM i Access for Windows cwbDQ\_GetKeyLen command.

## 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.

### **CWBDQ\_INVALID\_DATA\_HANDLE**

Invalid data handle.

## Usage

None

### **cwbDQ\_GetRetDataLen:**

Use the IBM i Access for Windows `cwbDQ_GetRetDataLen` command.

## 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.

### **CWBDQ\_INVALID\_DATA\_HANDLE**

Invalid data handle.

## Usage

None

### **cwbDQ\_GetRetKey:**

Use the IBM i Access for Windows cwbDQ\_GetRetKey command.

## 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:**

Use the IBM i Access for Windows cwbDQ\_GetRetKeyLen command.

## 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:**

Use the IBM i Access for Windows `cwbDQ_GetSearchOrder` command.

## 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\_GetSenderInfo:**

Use the IBM i Access for Windows cwbDQ\_GetSenderInfo command.

## 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\_SetConvert:**

Use the IBM i Access for Windows cwbDQ\_SetConvert command.

## 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.

### **CWBBDQ\_INVALID\_DATA\_HANDLE**

Invalid data handle.

## Usage

None

### **cwbDQ\_SetData:**

Use the IBM i Access for Windows cwbDQ\_SetData command.

## 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 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

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:**

Use the IBM i Access for Windows `cwbDQ_SetDataAddr` command.

## 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\_SetKey:**

Use the IBM i Access for Windows `cwbDQ_SetKey` command.

## 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\_SetSearchOrder:**

Use the IBM i Access for Windows cwbDQ\_SetSearchOrder command.

## 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

## Example: Using Data Queues APIs

The following example illustrates using IBM i data queues APIs.

```

// Sample Data Queues application

#ifdef UNICODE
    #define _UNICODE
    #define CWB_UNICODE
#endif
#include <windows.h>

// Include the necessary DQ Classes
#include <stdlib.h>
#include <iostream>
#include "cwbdq.h"

using namespace std;
/*****/

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;

    // obtain a handle to the system
    cwbcO_SysHandle system = NULL;
    if(cwbcO_CreateSystem( TEXT("SYSNAMEXXX"),&system) != 0)
        return;

    // Create the data queue DTAQ in library QGPL on system SYS1
    if ( cwbdQ_CreateEx(system,
                        TEXT("DTAQX"),
                        TEXT("QGPL"),
                        queueAttributes,
                        NULL) != 0 )
        return;

    // Delete the attributes
    if ( cwbdQ_DeleteAttr( queueAttributes ) != 0 )
        return;

    // Open the data queue
    if ( cwbdQ_OpenEx(system,
                     TEXT("DTAQ"),
                     TEXT("QGPL"),
                     &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;
}

```

## IBM i Access for Windows data transformation and National Language Support (NLS) APIs

Use Data Transformation and National Language Support (NLS) APIs to enable your applications to transform IBM i Access for Windows data.

### IBM i Access for Windows data transformation APIs

IBM i Access for Windows **data transformation** application programming interfaces (APIs) enable your client/server applications to transform IBM i numeric data between the system and the PC formats. Transformation may be required when you send and receive IBM i numeric data to and from the system. Data transformation APIs support transformation of many numeric formats.

### IBM i Access for Windows data transformation APIs required files:

Header file	Import library	Dynamic Link Library
cwbdtd.h	cwbdapi.lib	cwbdtd.dll

### Programmer's Toolkit:

The Programmer's Toolkit provides data transformation documentation, access to the cwbdtd.h header file, and links to sample programs. To access this information, open the Programmer's Toolkit and select **Data Manipulation** → **C/C++ APIs**.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

### Related reference

"IBM i name formats for connection APIs" on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

"OEM, ANSI, and Unicode considerations" on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

### IBM i Access for Windows data transformation API list:

The following IBM i Access for Windows data transformation APIs are listed alphabetically.

**Note:** IBM i 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.

*cwbDT\_ASCII11ToBin4*:

Use the IBM i Access for Windows `cwbDT_ASCII11ToBin4` command.

### 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 IBM i integer format.

### Unicode version

`cwbDT_Wide11ToBin4`

### 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 IBM i format that the system uses and is the opposite of the format that is used by the Intel<sup>®</sup> 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:*

Use the IBM i Access for Windows `cwbDT_ASCII6ToBin2` command.

## 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 IBM i integer format.

## Unicode version

`cwbDT_Wide6ToBin2`

## 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 IBM i format that the system 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:*

Use the IBM i Access for Windows `cwbDT_ASCIIpackedToPacked` command.

## Purpose

Translates data from ASCII packed format to packed decimal. This function can be used for translating data from ASCII files to the IBM i format

## Unicode version

None.

## 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:*

Use the IBM i Access for Windows `cwbDT_ASCIItoHex` command.

## Purpose

Translates data from ASCII (hex representation) to binary. One byte is stored in the target for each two bytes in the source.

## Unicode version

`cwbDT_WideToHex`

## 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:*

Use the IBM i Access for Windows `cwbDT_ASCIItoPacked` command.

## Purpose

Translates ASCII numeric data to packed decimal format. This function can be used for translating ASCII text data for use on the IBM i platform.

## Unicode version

`cwbDT_WideToPacked`

## 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:*

Use the IBM i Access for Windows *cwbDT\_ASCIItoZoned* command.

## Purpose

Translates ASCII numeric data to EBCDIC zoned decimal format. This function can be used for translating ASCII text data for use on the IBM i platform.

## Unicode version

cwbDT\_WideToZoned

## 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:*

Use the IBM i Access for Windows `cwbDT_ASCIIzonedToZoned` command.

## 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 IBM i platform.

## Unicode version

None.

## 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:*

Use the IBM i Access for Windows `cwbDT_Bin2ToASCII6` command.

### **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 IBM i numeric data to ASCII.

### **Unicode version**

`cwbDT_Bin2ToWide6`

### **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 IBM i format that the system uses and is the opposite of the format used by the Intel x86 processes.

*cwbDT\_Bin2ToBin2:*

Use the IBM i Access for Windows `cwbDT_Bin2ToBin2` command.

### **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 IBM i format.

### **Unicode version**

None.

### **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:*

Use the IBM i Access for Windows `cwbDT_Bin4ToASCII11` command.

## 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 IBM i numeric data to ASCII.

## Unicode version

`cwbDT_Bin4ToWide11`

## 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 IBM i format that the system uses and is the opposite of the format used by the Intel x86 processors.

*cwbDT\_Bin4ToBin4:*

Use the IBM i Access for Windows *cwbDT\_Bin4ToBin4* command.

### **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 IBM i format.

### **Unicode version**

None.

### **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:*

Use the IBM i Access for Windows *cwbDT\_EBCDICToEBCDIC* command.

## Purpose

'Translates' (copies unless character value less than 0x40 is encountered) EBCDIC data to EBCDIC.

## Unicode version

None.

## 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:*

Use the IBM i Access for Windows *cwbDT\_HexToASCII* command.

## Purpose

Translates binary data to the ASCII hex representation. Two ASCII characters are stored in the target for each byte of source data.

## Unicode version

*cwbDT\_HexToWide*

## 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:*

Use the IBM i Access for Windows *cwbDT\_PackedToASCII* command.

## Purpose

Translates data from packed decimal format to ASCII numeric data. This function can be used for translating IBM i data from the system for use in ASCII text format.

## Unicode version

*cwbDT\_PackedToWide*

## 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:*

Use the IBM i Access for Windows *cwbDT\_PackedToASCIIPacked* command.

**Purpose**

Translates data from packed decimal format to ASCII packed format. This function can be used for translating IBM i data from the system for use in ASCII format.

**Unicode version**

None.

**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:*

Use the IBM i Access for Windows `cwbDT_PackedToPacked` command.

### **Purpose**

Translates packed decimal data to packed decimal. This function can be used for transferring IBM i data from the system to no-conversion files and back.

### **Unicode version**

None.

### **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:*

Use the IBM i Access for Windows `cwbDT_ZonedToASCII` command.

### **Purpose**

Translates EBCDIC zoned decimal data to ASCII numeric format. This function can be used for translating IBM i data from the system for use in ASCII text format.

### **Unicode version**

`cwbDT_ZonedToWide`

### **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:*

Use the IBM i Access for Windows `cwbDT_ZonedToASCIIZoned` command.

## Purpose

Translates data from EBCDIC zoned decimal format to ASCII zoned decimal format. This function can be used for translating IBM i data from the system for use in ASCII files.

## Unicode version

None.

## 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:*

Use the IBM i Access for Windows `cwbDT_ZonedToZoned` command.

## Purpose

Translates data from zoned decimal format to zoned decimal. This function can be used for translating IBM i data from the system for use in no-conversion files and vice-versa.

## Unicode version

None.

## 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:

This example illustrates using IBM i Access for Windows data transformation APIs.

```
/******  
/* Sample Data Transform Program using cwbDT_Bin4ToBin4 to reverse */  
/* the order of bytes in a 4-byte integer. */  
/******  
  
#include <iostream>  
using namespace std;  
#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 */
}

```

## IBM i Access for Windows National Language Support (NLS) APIs

National Language Support APIs enable your applications to get and save (query and change) the IBM i Access for Windows settings that are relevant to different language versions.

Through NLS, the IBM i Access for Windows product supports many national languages. NLS allows users to work on a system in the language of their choice. The support 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 IBM i functions use a common set of program code, regardless of which language you use on the system. For example, the IBM i program code on a U.S. English language version and the IBM i program code on a Spanish language version 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, Online help information, and messages. This means that you see *Help* for the description of the function key for Online 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 system to support more than one language on a single system.

You can add convenient functions into your IBM i 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 IBM i operating system, 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.

**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.

### IBM i Access for Windows NLS APIs required files:

NLS API type	Header file	Import library	Dynamic Link Library
General	cwbnl.h	cwbbapi.lib	cwbnl.dll
Conversion	cwbnlcnv.h		cwbc core.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**.

### Related reference

"IBM i name formats for connection APIs" on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

"OEM, ANSI, and Unicode considerations" on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

### Coded character sets:

The IBM i Access for Windows product uses character encoding schemes.

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 the IBM i Access Family 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 on the IBM i platform.
- PC Code pages are encoded based on ANSI X3.4, ASCII and usually used by IBM Personal Computers.

### IBM i Access for Windows general NLS APIs list:

Use IBM i Access for Windows general NLS APIs.

IBM i Access for Windows is translated into many languages. One or more of these languages can be installed on the personal computer. The following IBM i 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*:

Use the IBM i Access for Windows *cwbNL\_FindFirstLang* command.

### 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 errorHandle);
```

## Parameters

### **char \* mriBasePath - input**

| Pointer to the mriBasePath, for example C:\Program Files\IBM\ClientAccess. If NULL, the  
| mriBasePath of the IBM i Access for Windows 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 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\_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*:

Use the IBM i Access for Windows *cwbNL\_FindNextLang* command.

## 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 errorHandle);
```

## 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 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\_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:*

Use the IBM i Access for Windows `cwbNL_GetLang` command.

## 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`, for example `C:\Program Files\IBM\ClientAccess`. If `NULL`, the  
| `mriBasePath` of the IBM i Access for Windows 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*:

Use the IBM i Access for Windows `cwbNL_GetLangName` command.

## 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 ASCII 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*:

Use the IBM i Access for Windows `cwbNL_GetLangPath` command.

## 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`. If `NULL`, the  
| `mriBasePath` of the IBM i Access for Windows 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:*

Use the IBM i Access for Windows *cwbNL\_SaveLang* command.

### **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*



## IBM i Access for Windows conversion NLS APIs list:

This topic describes the IBM i Access for Windows conversion NLS APIs.

The following IBM i 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:*

Use the IBM i Access for Windows *cwbNL\_CCSIDToCodePage* command.

### Purpose

Map CCSIDs to code pages.

### Syntax

```
unsigned int CWB_ENTRY cwbNL_CCSIDToCodePage(  
    unsigned long  CCSID,  
    unsigned long *codePage,  
    cwbSV_ErrHandle errorHandler);
```

### Parameters

#### **unsigned long CCSID - input**

CCSID to convert to a code page.

#### **unsigned long \* codePage - output**

The resulting code page.

#### **cwbSV\_ErrHandle errorHandler - 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:*

Use the IBM i Access for Windows `cwbNL_CodePageToCCSID` command.

### 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:*

Use the IBM i Access for Windows `cwbNL_Convert` command.

### 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:*

Use the IBM i Access for Windows `cwbNL_ConvertCodePages` command.

## Comments

`cwbNL_ConvertCodePages` is no longer supported. See `cwbNL_ConvertCodePagesEx`.

*`cwbNL_ConvertCodePagesEx`:*

Use the IBM i Access for Windows `cwbNL_ConvertCodePagesEx` command.

## Purpose

Convert strings from one code page to another. This API combines the following three converter APIs for the default conversion:

- `cwbNL_CreateConverterEx`
- `cwbNL_Convert`
- `cwbNL_DeleteConverter`

## Syntax

```
unsigned int CWB_ENTRY cwbNL_ConvertCodePagesEx(  
    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 IBM i Access for Windows, or retrieved from the default system when needed. There may have been some problem communicating with the default 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:

<b>Value</b>	<b>Meaning</b>
CWBNL_CP_UNICODE_F200	UCS2 Version 1.1 UNICODE
CWBNL_CP_UNICODE	UCS2 Current Version UNICODE
CWBNL_CP_AS400	IBM i 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_CP_UTF16BE	UTF-16 (Big-Endian)
CWBNL_CP_UTF16LE	UTF-16 (Little-Endian)
CWBNL_CP_UTF16	CWBNL_CP_UTF16BE or CWBNL_CP_UTF16LE, depending on the platform
CWBNL_CP_UTF32BE	UTF-32 (Big-Endian)
CWBNL_CP_UTF32LE	UTF-32 (Little-Endian)
CWBNL_CP_UTF32	CWBNL_CP_UTF32BE or CWBNL_CP_UTF32LE, depending on the platform

*cwbNL\_CreateConverter:*

Use the IBM i Access for Windows `cwbNL_CreateConverter` command.

### Comments

`cwbNL_CreateConverter` is no longer supported. See `cwbNL_CreateConverterEx`.

### 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 IBM i Access for Windows, or retrieved from the default system when needed. There may have been some problem communicating with the default 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:

<b>Value</b>	<b>Meaning</b>
CWBNL_CP_UNICODE_F200	UCS2 Version 1.1 UNICODE
CWBNL_CP_UNICODE	UCS2 Current Version UNICODE
CWBNL_CP_AS400	IBM i 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_CP_UTF16BE	UTF-16 (Big-Endian)
CWBNL_CP_UTF16LE	UTF-16 (Little-Endian)
CWBNL_CP_UTF16	CWBNL_CP_UTF16BE or CWBNL_CP_UTF16LE, depending on the platform
CWBNL_CP_UTF32BE	UTF-32 (Big-Endian)
CWBNL_CP_UTF32LE	UTF-32 (Little-Endian)
CWBNL_CP_UTF32	CWBNL_CP_UTF32BE or CWBNL_CP_UTF32LE, depending on the platform

Instead of calling `cwbnl_ConvertCodePagesEx` multiple times with the same code pages:

- `cwbnl_ConvertCodePagesEx(850, 500, ...);`
- `cwbnl_ConvertCodePagesEx(850, 500, ...);`
- `cwbnl_ConvertCodePagesEx(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\_CreateConverterEx:*

Use the IBM i Access for Windows `cwbNL_CreateConverterEx` command.

## Purpose

Create a `cwbNL_Converter` to be used on subsequent calls to `cwbNL_Convert()`.

## Syntax

```
unsigned int CWB_ENTRY cwbNL_CreateConverterEx(
    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 the IBM i Access for Windows product, or retrieved from the default system when needed. There may have been some problem communicating with the default 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
CWBNL_CP_UNICODE_F200	UCS2 Version 1.1 UNICODE
CWBNL_CP_UNICODE	UCS2 Current Version UNICODE
CWBNL_CP_AS400	IBM i 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_CP_UTF16BE	UTF-16 (Big-Endian)
CWBNL_CP_UTF16LE	UTF-16 (Little-Endian)
CWBNL_CP_UTF16	CWBNL_CP_UTF16BE or CWBNL_CP_UTF16LE, depending on the platform
CWBNL_CP_UTF32BE	UTF-32 (Big-Endian)
CWBNL_CP_UTF32LE	UTF-34 (Little-Endian)
CWBNL_CP_UTF32	CWBNL_CP_UTF32BE or CWBNL_CP_UTF32LE, depending on the platform

Instead of calling `cwbnl_ConvertCodePagesEx` multiple times with the same code pages:

- `cwbnl_ConvertCodePagesEx(850, 500, ...);`
- `cwbnl_ConvertCodePagesEx(850, 500, ...);`
- `cwbnl_ConvertCodePagesEx(850, 500, ...);`

It is more efficient to create a converter and use it multiple times:

- `cwbNL_CreateConverterEx(850, 500, &conv, ...);`
- `cwbNL_Convert(conv, ...);`
- `cwbNL_Convert(conv, ...);`
- `cwbNL_Convert(conv, ...);`
- `cwbNL_DeleteConverter(conv, ...);`

*cwbNL\_DeleteConverter:*

Use the IBM i Access for Windows `cwbNL_DeleteConverter` command.

### **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:*

Use the IBM i Access for Windows `cwbNL_GetCodePage` command.

### **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 IBM i Access Family 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:*

Use the IBM i Access for Windows `cwbNL_GetANSICodePage` command.

## 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 IBM i Access Family 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:*

Use the IBM i Access for Windows *cwbNL\_GetHostCCSID* command.

## 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 IBM i Access for Windows **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.

### **IBM i Access for Windows dialog-box NLS API list:**

IBM i Access for Windows dialog-box NLS APIs are interfaces that are used to manipulate the translatable text within dialog boxes.

The following IBM i 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

### **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:*

Use the IBM i Access for Windows *cwbNL\_CalcControlGrowthXY* command.

### **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:*

Use the IBM i Access for Windows *cwbNL\_CalcDialogGrowthXY* command.

## 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:*

Use the IBM i Access for Windows *cwbNL\_GrowControlXY* command.

## 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:*

Use the IBM i Access for Windows *cwbNL\_GrowDialogXY* command.

## 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:*

Use the IBM i Access for Windows *cwbNL\_LoadDialogStrings* command.

## 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:*

Use the IBM i Access for Windows *cwbNL\_LoadMenu* command.

**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:*

Use the IBM i Access for Windows *cwbNL\_LoadMenuStrings* command.

## 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:*

Use the IBM i Access for Windows *cwbNL\_SizeDialog* command.

## 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: IBM i Access for Windows NLS APIs:

This example illustrates using IBM i 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 "CWBNL CNV.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, (char*)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 nmberrs;
unsigned long posoferr;
unsigned long rqdlen;
unsigned long ccid;

/* 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_OK) {
    if (NLrc == CWB_BUFFER_OVERFLOW)
        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_OK) {
    if (NLrc == CWB_BUFFER_OVERFLOW)
        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_OK) {
    if (NLrc == CWB_BUFFER_OVERFLOW)
        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_OK) {
    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_CreateConverterEx                        */
/*     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_ConvertCodePagesEx(codepage, trgtpage, srcLen,
    trgtlen, srcbuf, trgtbuf, &nmbrrrs, &posoferr, &rqdlen,
    errhandle);
if (NLrc != CWB_OK) {
    resolveErr(errhandle);
    printf("number of errors detected is %u.\n", nmbrrrs);
    printf("location of first error is %u.\n", posoferr);
}
printf("ConvertCodePagesEx API returned %s.\n", trgtbuf);

/* Map a code page to the corresponding CCSID.          */
NLrc = cwbNL_CodePageToCCSID(codepage, &ccsid, errhandle);
if (NLrc != CWB_OK) {
    resolveErr(errhandle);
}
printf("CodePageToCCSID returned %u.\n", ccsid);

cwbSV_DeleteErrHandle(errhandle);
}

```

## IBM i Access for Windows Directory Update APIs

Specify PC directory updates using the IBM i Access for Windows Directory Update function.

### IBM i Access for Windows Directory Update C/C++ APIs:

IBM i 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 IBM i Access for Windows Directory Update function.


**Note:** IBM i 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.

IBM i 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 IBM i Access for Windows client does not include support for network drives or for universal naming conventions. This now is provided by the **IBM i NetServer** function. IBM i Access mapped Network drives should be mapped by using IBM i NetServer support. Set up the IBM i NetServer that comes with IBM i in order to perform IBM i file serving.

**NetServer information resources:**

- IBM i NetServer topic of the IBM i Information Center
- IBM IBM i NetServer Home Page 

**IBM i 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**.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

**Related reference**

“Directory Update APIs return codes” on page 26

There are IBM i Access for Windows Directory Update API return codes.

“IBM i name formats for connection APIs” on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

“OEM, ANSI, and Unicode considerations” on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

**Typical use of IBM i Access for Windows Directory Update APIs**

IBM i 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, IBM i files might contain customer names and addresses. The IBM i files 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 IBM i files 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 IBM i Access for Windows 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 IBM i Access for Windows 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 214 for more information. Package files are added to update entries by using the “cwbUP\_AddPackageFile” on page 220 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 222 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 \_declspec(dllexport) cwbUP\_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** used by the IBM i Access for Windows product 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

For a Directory Update C/C++ sample program, you can go to the IBM i Access for Windows Programmer’s Toolkit - Directory Update Web page.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.



Go to 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 IBM i Access for Windows User's Guide for more information.

## Directory Update: Create and delete APIs

The following IBM i Access for Windows Directory Update are used to create and delete an update entry. The APIs are listed alphabetically.

**Note:** It is essential that 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.

### **cwbUP\_CreateUpdateEntry:**

Use the IBM i Access for Windows `cwbUP_CreateUpdateEntry` command.

#### **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:**

Use the IBM i Access for Windows `cwbUP_DeleteEntry` command.

**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.

**Directory Update: Access APIs**

The following IBM i Access for Windows Directory Update are used to obtain access to an update entry. The APIs are listed alphabetically.

**Note:** It is essential that 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.

### **cwbUP\_FindEntry:**

Use the IBM i Access for Windows `cwbUP_FindEntry` command.

### **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\_FreeLock:**

Use the IBM i Access for Windows `cwbUP_FreeLock` command.

### **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\_GetEntryHandle:**

Use the IBM i Access for Windows `cwbUP_GetEntryHandle` command.

### **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.

## Directory Update: Free Resources APIs

The following IBM i Access for Windows Directory Update APIs are used to free resources that are associated with an entry handle. The APIs are listed alphabetically.

**Note:** It is essential that 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.

### **cwbUP\_FreeEntryHandle:**

Use the IBM i Access for Windows cwbUP\_FreeEntryHandle command.

## Purpose

Frees an entry handle and all resources with which is 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.

## Directory Update: Change APIs

The following IBM i Access for Windows Directory Update APIs are used to change an update entry. The APIs are listed alphabetically.

**Note:** It is essential that 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.

### **cwbUP\_AddPackageFile:**

Use the IBM i Access for Windows `cwbUP_AddPackageFile` command.

## 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\_RemovePackageFile:**

Use the IBM i Access for Windows cwbUP\_RemovePackageFile command.

**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:**

Use the IBM i Access for Windows cwbUP\_SetCallbackDLL command.

## 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:**

Use the IBM i Access for Windows cwbUP\_SetDescription command.

## 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:**

Use the IBM i Access for Windows `cwbUP_SetEntryAttributes` command.

## 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:**

Use the IBM i Access for Windows `cwbUP_SetSourcePath` command.

## **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:**

Use the IBM i Access for Windows `cwbUP_SetTargetPath` command.

## 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

**Directory Update: Information APIs**

The following IBM i Access for Windows Directory Update APIs are used to obtain information from an update entry and to retrieve general Directory Update information. The APIs are listed alphabetically.

**Note:** It is essential that 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.

**cwbUP\_GetCallbackDLL:**

Use the IBM i Access for Windows `cwbUP_GetCallbackDLL` command.

**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 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 \* 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 bufferLength - 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:**

Use the IBM i Access for Windows cwbUP\_GetDescription command.

**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:**

Use the IBM i Access for Windows `cwbUP_GetEntryAttributes` command.

**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\_GetLockHolderName:**

Use the IBM i Access for Windows `cwbUP_GetLockHolderName` command.

### **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:**

Use the IBM i Access for Windows `cwbUP_GetSourcePath` command.

### **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:**

Use the IBM i Access for Windows `cwbUP_GetTargetPath` command.

## Purpose

Gets the target path of the update entry.

## Syntax

```
unsigned int CWB_ENTRY cwbUP_GetTargetPath(  
    cwbUP_EntryHandle entryHandle,  
    char *entryTarget,  
    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 \* entryTarget - input/output**

Pointer to a buffer that will receive the target 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 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

**IBM i Access for Windows PC5250 emulation APIs**

The IBM i Access for Windows PC5250 emulator provides desktop users with a graphical user interface for existing system applications. PC5250 allows users to easily and transparently interact with IBM i stored data and applications.

PC5250 provides C/C++ application programming interfaces (APIs) for enabling workstation programs to interact with IBM i host systems.

**IBM i Access for Windows PC5250 C/C++ APIs:****Emulator high-level language API (EHLAPI)**

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.

**IBM i Access for Windows emulation APIs required files:**

Emulation interface	Header file	Import library	Dynamic Link Library
Standard HLLAPI	hapi_c.h	pascal32.lib	pcshll.dll pcshll32.dll
Enhanced HLLAPI	ehlapi32.h	ehlapi32.lib	ehlapi32.dll
Windows EHLAPI	whllapi.h	whllapi.lib whlapi32.lib	whllapi.dll whlapi32.dll

Emulation interface	Header file	Import library	Dynamic Link Library
HACL interface	eclall.hpp	pcseclva.lib pcseclvc.lib	pcseclva.dll pcseclvc.dll
PCSAPI interface	pcsapi.h	pcscal32.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**.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

### System Objects APIs for IBM i Access for Windows

System objects for IBM i Access for Windows application programming interfaces (APIs) allow you to work with print-related objects that are on the system. These APIs make it possible to work with IBM i spooled files, writer jobs, output queues, printers, and more.

By using System 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 IBM i operating systems. This includes holding, releasing, changing attributes of, deleting, sending, retrieving and answering messages for the spooled files.

### System Objects APIs for IBM i 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 System 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 **IBM i Operations** → **C/C++ APIs**.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

## Related reference

“System Object APIs return codes” on page 28

There are IBM i Access for Windows system object API return codes.

“IBM i name formats for connection APIs” on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

“OEM, ANSI, and Unicode considerations” on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

## System 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.

### Advanced Function Printing:

Use this API with the IBM i Access for Windows product.

**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:

Use this API with the IBM i Access for Windows product.

**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:

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_ALWDRTPT

**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:

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_BKOVRLLIB

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_BKOVRL\_ACR

**ID** 0x0016

**Type** float

**Description**

The offset across from the point of origin where the overlay is printed.

**Back Overlay Offset Down:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_BKOVRL\_DWN

**ID** 0x0015

**Type** float

**Description**

The offset down from the point of origin where the overlay is printed.

**Characters per Inch:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_CPI

**ID** 0x0017

**Type** float

**Description**

The number of characters per horizontal inch.

**Code Page:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_COPIESLEFT

**ID** 0x001D

**Type** long

**Description**

The remaining number of copies to be produced for this spooled file.

**Current page:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_CURPAGE

**ID** 0x001E

**Type** long

**Description**

Current page that is being written by the writer job.

**Data Format:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DATAFORMAT

**ID** 0x001F

**Type** char[11]

**Description**

Data format. Valid values are \*RCDDATA, \*ALLDATA.

**Data Queue Library Name:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DATAQUELIB

**ID** 0x0020

**Type** char[11]

**Description**

The name of the library that contains the data queue.

**Data Queue Name:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DBCSEXTENS

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DFR\_WRITE

**ID** 0x0023

**Type** char[11]

**Description**

Whether print data is held in system buffers before being sent to the printer. Valid values are \*YES, \*NO.

**Degree of Page Rotation:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_PAGRRT

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DELETESPLF

**ID** 0x0097

**Type** char[11]

**Description**

Delete the spooled file after sending? Valid values are \*NO and \*YES.

**Destination Option:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DESTINATION

**ID** 0x0025

**Type** char[11]

**Description**

Destination type. Valid values are \*OTHER, \*AS400, \*PSF2.

**Device Class:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DEVCLASS

**ID** 0x0026

**Type** char[11]

**Description**

The device class.

**Device Model:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DEVMODEL

**ID** 0x0027

**Type** char[11]

**Description**

The model number of the device.

**Device Type:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DEVTYPE

**ID** 0x0028

**Type** char[11]

**Description**

The device type.

**Display any File:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_HOLD

**ID** 0x0039

**Type** char[11]

**Description**

Whether the spooled file is held. Valid values are \*YES, \*NO.

**Initialize the writer:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_INTERNETADDR

**ID** 0x0094

**Type** char[16]

**Description**

The internet address of the receiving system.

**Job Name:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_JOBNAME

**ID** 0x003B

**Type** char[11]

**Description**

The name of the job that created the spooled file.

**Job Number:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_JOBNUMBER

**ID** 0x003C

**Type** char[7]

**Description**

The number of the job that created the spooled file.

**Job Separators:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_USER

**ID** 0x003E

**Type** char[11]

**Description**

The name of the user that created the spooled file.

**Last Page Printed:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_LIBRARY

**ID** 0x000F

**Type** char[11]

**Description**

The name of the library.

**Lines Per Inch:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_LPI

**ID** 0x0040

**Type** float

**Description**

The number of lines per vertical inch in the spooled file.

**Manufacturer Type and Model:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**



Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_MESSAGEID

**ID** 0x0093

**Type** char[8]

**Description**

The message ID.

**Message Queue Library Name:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_MSGQUELIB

**ID** 0x0044

**Type** char[11]

**Description**

The name of the library that contains the message queue.

**Message Queue:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_NUMWRITERS

**ID** 0x0091

**Type** long

**Description**

The number of writer jobs started to the output queue.

**Object Extended Attribute:**

Use this API with the IBM i Access for Windows product.

**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 the WRKOBJ and DSPOBJD IBM i commands. The title on an IBM i 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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_OPENCMDS

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_OUTQUELIB

**ID** 0x0049

**Type** char[11]

**Description**

The name of the library that contains the output queue.

**Output Queue Name:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_OUTQUEUE

**ID** 0x004A

**Type** char[11]

**Description**

The name of the output queue.

**Output Queue Status:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_OUTQUESTS

**ID** 0x004B

**Type** char[11]

**Description**

The status of the output queue. Valid values are RELEASED, HELD.

**Overflow Line Number:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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 system defines them.

**Point Size:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DUPLEX

**ID** 0x0055

**Type** char[11]

**Description**

How the information prints. Valid values are \*FORMDF, \*NO, \*YES, \*TUMBLE.

**Print Quality:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_PRTSEQUENCE

**ID** 0x0057

**Type** char[11]

**Description**

Print sequence. Valid values are \*NEXT.

**Print Text:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_PRINTER

**ID** 0x0059

**Type** char[11]

**Description**

The name of the printer device.

**Printer Device Type:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_PRTRFILELIB

**ID** 0x005B

**Type** char[11]

**Description**

The name of the library that contains the printer file.

**Printer File Name:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_PRTRFILE

**ID** 0x005C

**Type** char[11]

**Description**

The name of the printer file.

**Printer Queue:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_RMTPRTQ

**ID** 0x005D

**Type** char[129]

**Description**

The name of the destination printer queue when sending spooled files via SNDTCPSPLF (LPR).

**Record Length:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_RECLENGTH

**ID** 0x005F

**Type** long

**Description**

Record length.

**Remote System:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_RMTSYSTEM

**ID** 0x0060

**Type** char[256]

**Description**

Remote system name. Valid special values include \*INTNETADR.

**Replace Unprintable Characters:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_RPLCHAR

**ID** 0x0062

**Type** char[2]

**Description**

The character that replaces any unprintable characters.

**Resource library name:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_RSCNAME

**ID** 0x00AF

**Type** char[11]



**Description**

The name of the external AFP resource.

**Resource object type:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_SEEKOFF

**ID** 0x007E

**Type** long

**Description**

Seek offset. Allows both positive and negative values relative to the seek origin.

**Seek Origin:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_SENDPTY

**ID** 0x0065

**Type** char[11]

**Description**

Send priority. Valid values are \*NORMAL, \*HIGH.

**Separator page:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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 encoded as -1, and the value \*FORMDF is encoded as -2.

**Spool SCS:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_SPOOLFILE

**ID** 0x0068

**Type** char[11]

**Description**

The name of the spooled file.

**Spooled File Number:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_SPLFNUM

**ID** 0x0069

**Type** long

**Description**

The spooled file number.

**Spooled File Status:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_DESCRIPTION

**ID** 0x006D

**Type** [51]

**Description**

Text to describe an instance of an IBM i object.

**Time File Opened:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_PAGES

**ID** 0x006F

**Type** long

**Description**

The number of pages that are contained in a spooled file.

**Transform SCS to ASCII:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_USERCMT

**ID** 0x0073

**Type** char[101]

**Description**

The 100 characters of user-specified comment that describe the spooled file.

**User Data:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_USRDFNDTA

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_USRDFNOBJTYP

**ID** 0x00A6

**Type** char[11]

**Description**

User defined object type pertaining to the user defined object.

**User defined option(s):**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_USRDRVPGM

**ID** 0x00AB

**Type** char[11]

**Description**

User defined program name that processes spool files.

**User ID:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_TOUSERID

**ID** 0x0075

**Type** char[9]

**Description**

User ID to which the spooled file is sent.

**User ID Address:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_TOADDRESS

**ID** 0x0076

**Type** char[9]

**Description**

Address of user to whom the spooled file is sent.

**User transform program library:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_USRTFMPGMLIB

**ID** 0x00A7

**Type** char[11]

**Description**

User defined library search for transform program.

**User transform program name:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_WSCUSTMOBJ



**ID** 0x0095

**Type** char[11]

**Description**

The name of the workstation customizing object.

**Workstation Customizing Object Library:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_WSCUSTMOBJL

**ID** 0x0096

**Type** char[11]

**Description**

the name of the library that contains the workstation customizing object.

**Writer Job Name:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_WRITER

**ID** 0x0079

**Type** char[11]

**Description**

The name of the writer job.

**Writer Job Number:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_WTRJOBNUM

**ID** 0x007A

**Type** char[7]

**Description**

The writer job number.

**Writer Job Status:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_WTRJOBUSER

**ID** 0x007C

**Type** char[11]

**Description**

The name of the user that started the writer job.

**Writer Starting Page:**

Use this API with the IBM i Access for Windows product.

**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:**

The follow list is for object attributes for the network print server when using the IBM i Access for Windows product.

*NPS Attribute Default Value:*

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_ATTRDEFAULT

**ID** 0x0083

**Type** dynamic

**Description**

Default value for the attribute.

*NPS Attribute High Limit:*

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_ATTRMAX

**ID** 0x0084

**Type** dynamic

**Description**

High limit of the attribute value.

*NPS Attribute ID:*

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_ATTRID

**ID** 0x0085

**Type** long

**Description**

ID of the attribute.

*NPS Attribute Low Limit:*

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_ATTRMIN

**ID** 0x0086

**Type** dynamic

**Description**

Low limit of the attribute value.

*NPS Attribute Possible Value:*

Use this API with the IBM i Access for Windows product.

**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:*

Use this API with the IBM i Access for Windows product.

**Key** CWBOBJ\_KEY\_ATTRDESCRIPT

**ID** 0x0088

**Type** char(\*)

**Description**

Text description that provides a name for the attribute.

*NPS Attribute Type:*

Use this API with the IBM i Access for Windows product.

**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:*

Use this API with the IBM i Access for Windows product.

**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:*

Use this API with the IBM i Access for Windows product.

**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:*

Use this API with the IBM i Access for Windows product.

**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:*

Use this API with the IBM i Access for Windows product.

**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)

## List APIs for IBM i Access for Windows

The following IBM i Access for Windows APIs pertain to List objects. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwbOBJ\_CloseList:**

Use this API with the IBM i Access for Windows product.

#### **Purpose**

Closes an opened list.

#### **Syntax**

```
unsigned int CWB_ENTRY cwbOBJ_CloseList(  
    cwbOBJ_ListHandle listHandle,  
    cwbSV_ErrHandle errorHandler);
```

## 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\_OK**

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\_CreateListHandle:**

Use this API with the IBM i Access for Windows product.

## 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 deallocated 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 ASCII 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\_OK**

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\_DeleteListHandle:**

Use this API with the IBM i Access for Windows product.

**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\_OK**

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\_GetListSize:**

Use this API with the IBM i Access for Windows product.

### **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\_OK**

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\_OpenList:**

Use this API with the IBM i Access for Windows product.

**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,
                        cwboSV_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

**cwboSV\_ErrHandle errorHandle - output**

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwboSV\_CreateErrHandle() API. The messages may be retrieved through the cwboSV\_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.

**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\_ResetListAttrsToRetrieve:**

Use this API with the IBM i Access for Windows product.

**Purpose**

Resets the list attributes to retrieve information to its default list.

**Syntax**

```
unsigned int CWB_ENTRY  cwbOBJ_ResetListAttrsToRetrieve(  
                        cwbOBJ_ListHandle  listHandle,  
                        cwbSV_ErrHandle    errorHandle);
```

**Parameters****cwbOBJ\_ListHandle listHandle - input**

List handle to 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\_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:**

Use this API with the IBM i Access for Windows product.

**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\_OK**

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\_SetListAttrsToRetrieve:**

Use this API with the IBM i Access for Windows product.

## 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 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\_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:**

Use this API with the IBM i Access for Windows product.

**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 errorHandler);
```

**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\_OK**

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 OUTQUEUE filter is not \*ALL. "" is the default if the OUTQU filter is set to \*ALL.
  - CWBOBJ\_KEY\_OUTQUEUE  
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 o 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 o 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 active 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_SetListFilterWithSpIF` 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 (for example, `SetListFilterWithSpIF`). 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:**

Use this API with the IBM i Access for Windows product.

### **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\_OK**

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.

## **IBM i Access for Windows Object APIs**

The following IBM i Access for Windows APIs pertain to Objects. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwbOBJ\_CopyObjHandle:**

Use this API with the IBM i Access for Windows product.

### **Purpose**

Creates a duplicate handle to an object. Use this API to get another handle to the same IBM i 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\_OK**

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 close this API allows you to do that. `cwbOBJ_DeleteObjHandle()` must be called to release resources for this handle.

### **cwbOBJ\_DeleteObjHandle:**

Use this API with the IBM i Access for Windows product.

## 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\_OK**

Successful completion.

### **CWB\_NOT\_ENOUGH\_MEMORY**

Insufficient memory.

### **CWB\_INVALID\_HANDLE**

Handle is not an allocated object handle.



## Usage

None

### **cwbOBJ\_GetObjAttr:**

Use this API with the IBM i Access for Windows product.

## 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\_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\_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_BKOVRLLIB	- Back overlay library name
CWBOBJ_KEY_BKOVRLAY	- Back overlay name
CWBOBJ_KEY_BKOVLD_ACR	- Back overlay offset across
CWBOBJ_KEY_BKOVLD_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_FTOVLD_ACR	- Front overlay offset across
CWBOBJ_KEY_FTOVLD_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_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\_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\_USRDFNDA - 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\_USRDFNDA - 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\_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\_KEY\_USRTFMPGLIB- 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\_BKMGN\_ACR - back margin across  
 CWBOBJ\_KEY\_BKMGN\_DWN - back margin down  
 CWBOBJ\_KEY\_BKOVRLLIB - 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\_DBCSEXTENSN - 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:**

Use this API with the IBM i Access for Windows product.

#### **Purpose**

Get several attributes of an object.

#### **Syntax**

```
unsigned int CWB_ENTRY cwBOBJ_GetObjAttrs(
    cwBOBJ_ObjHandle    objectHandle,
    unsigned long        numAttrs,
    cwBOBJ_GetObjAttrParms *getAttrParms,
    cwSV_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\_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\_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:**

Use this API with the IBM i Access for Windows product.

## Purpose

Get list object. This call gets a handle to an object in an opened list. The handle returned must be released with 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\_OK**

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:**

Use this API with the IBM i Access for Windows product.

## 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\_OK**

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:**

Use this API with the IBM i Access for Windows product.

## Purpose

Get the id of an object. This is the data the 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\_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.

## Usage

None

### **cwbOBJ\_RefreshObj:**

Use this API with the IBM i Access for Windows product.

## Purpose

Refreshes the object with the latest IBM i information. 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\_OK**

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_OK)
{
    u1RC = cwbOBJ_RefreshObj(splFileHandle);
    .....
    get attributes for object
    .....
    u1RC = cwbOBJ_DeleteObjHandle(splFileHandle);
}
```

### **cwbOBJ\_SetObjAttrs:**

Use this API with the IBM i Access for Windows product.

## Purpose

Change the attributes of the object on the server.

## Syntax

```
unsigned int CWB_ENTRY cwboBJ_SetObjAttrs(  
    cwboBJ_ObjHandle  objectHandle,  
    cwboBJ_ParmHandle parmListHandle,  
    cwboSV_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.

### **cwboSV\_ErrHandle errorHandle - output**

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwboSV\_CreateErrHandle() API. The messages may be retrieved through the cwboSV\_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.

### **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\_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\_OUTPTY - Output priority
  - CWBOBJ\_KEY\_OUTQUELIB - Output queue library name
  - CWBOBJ\_KEY\_OUTQUE - Output queue
  - CWBOBJ\_KEY\_MULTIUPT - 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\_BKMGN\_ACR - Back margin offset across  
 CWBOBJ\_KEY\_BKMGN\_DWN - Back margin offset down  
 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  
 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\_DBCSEXTENSN - 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\_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\_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\_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\_OUTQ (output queues):
- CWBOBJ\_LIST\_PRTD (printer devices):
- CWBOBJ\_LIST\_WTR (writers):
- CWBOBJ\_LIST\_LIB (libraries):
  - NONE

## IBM i Access for Windows Parameter object APIs

The following IBM i Access for Windows APIs pertain to Parameter objects. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwBOBJ\_CopyParmObjHandle:**

Use this API with the IBM i Access for Windows product.

#### **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,
    cwSV_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.

##### **cwSV\_ErrHandle errorHandle - output**

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwSV\_CreateErrHandle() API. The messages may be retrieved through the cwSV\_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\_CreateParmObjHandle:**

Use this API with the IBM i Access for Windows product.

### **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\_OK**

Successful completion.

#### **CWB\_NOT\_ENOUGH\_MEMORY**

Insufficient memory.

### **Usage**

The `cwbOBJ_DeleteParmObjectHandle` API must be called to free resources allocated by this call.

### **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\_OK**

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\_GetParameter:**

Use this API with the IBM i Access for Windows product.

## 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\_SetParameter:**

Use this API with the IBM i Access for Windows product.

**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 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\_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

**IBM i Access for Windows Writer job APIs**

The following IBM i Access for Windows APIs pertain to Writer job. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

**cwbOBJ\_EndWriter:**

Use this API with the IBM i Access for Windows product.

**Purpose**

Ends an IBM i writer job.

**Syntax**

```
unsigned int CWB_ENTRY cwbOBJ_EndWriter(
    cwbOBJ_ObjHandle  writerHandle,
    cwbOBJ_ParmHandle *parmListHandle,
    cwbSV_ErrHandle  errorHandler);
```

**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\_OK**

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 `pParmListHandle` 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\_StartWriter:**

Use this API with the IBM i Access for Windows product.

### **Purpose**

Starts an IBM i 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\_OUTQUEUE 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\_OK**

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 IBM i STRPRTWTR command. 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

## IBM i Access for Windows output queues APIs

The following IBM i Access for Windows APIs pertain to Output queues. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwbOBJ\_HoldOutputQueue:**

Use this API with the IBM i Access for Windows product.

#### **Purpose**

Holds an IBM i 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\_OK**

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\_PurgeOutputQueue:**

Use this API with the IBM i Access for Windows product.

#### **Purpose**

Purges spooled files on an IBM i 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\_OK**

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\_ReleaseOutputQueue:**

Use this API with the IBM i Access for Windows product.

## Purpose

Releases an IBM i 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\_OK**

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

## IBM i Access for Windows AFP resource APIs

The following IBM i Access for Windows APIs pertain to AFP resources. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwbOBJ\_CloseResource:**

Use this API with the IBM i Access for Windows product.

## Purpose

Closes an AFP Resource object that was previously opened for reading.

## Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_CloseResource(  
                        cwbOBJ_ObjHandle  resourceHandle,  
                        cwbSV_ErrHandle   errorHandle);
```

## Parameters

### **cwbOBJ\_ObjHandle resourceHandle - input**

Handle of the resource 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\_OK**

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\_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\_OpenResourceForSplF() 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\_CreateResourceHandle:**

Use this API with the IBM i Access for Windows product.

## 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 ASCIIZ string.

**const char \*resourceName - input**

Pointer to the name of the AFP resource.

**const char \*resourceLibrary - input**

Pointer to the name of the IBM i 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\_DisplayResource:**

Use this API with the IBM i Access for Windows product.

**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 ASCIIZ 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\_DSPSPFL\_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\_OpenResource:**

Use this API with the IBM i Access for Windows product.

### Purpose

Opens an AFP resource object for reading.

### Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_OpenResource(  
                        cwbOBJ_ObjHandle  resourceHandle,  
                        cwbSV_ErrHandle   errorHandler);
```

### Parameters

#### **cwbOBJ\_ObjHandle resourceHandle - input**

Handle of the AFP resource file to be opened for reading.

#### **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\_INVALID\_HANDLE**

Handle is not valid resource handle.

#### **CWBOBJ\_RC\_HOST\_ERROR**

Host error occurred. Text may be in errorHandler.

#### **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\_OpenResourceForSp1F:**

Use this API with the IBM i Access for Windows product.

### 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_OpenResourceForSp1F(  
                        cwbOBJ_ObjHandle  sp1FHandle,  
                        const char        *resourceName,
```

```

const char      *resourceLibrary,
unsigned long   resourceType,
const char     *reserved,
cwbOBJ_ObjHandle *resourceHandle,
cwbSV_ErrHandle errorHandle);

```

## Parameters

### **cwbOBJ\_ObjHandle splFHandle - input**

Handle of the spooled file that is already opened for reading and that the resource is opened against. The same system conversation (and same system instance of the network print server program) is used for reading the resource and spooled file.

### **const char \*resourceName - input**

Pointer to the name of the AFP Resource in an ASCII string.

### **const char \*resourceLibrary - input**

Optional, may be NULL. Pointer to the IBM i library of the AFP Resource in an ASCII 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\_PAGEDDEF

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\_OK**

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\_SPLFNOTOPEN**

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\_ReadResource:**

Use this API with the IBM i Access for Windows product.

**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,
                        cwboSV_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.

**cwboSV\_ErrHandle errorHandle - output**

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwboSV\_CreateErrHandle() API. The messages may be retrieved through the cwboSV\_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 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_OpenResourceForSplF()` 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\_SeekResource:**

Use this API with the IBM i Access for Windows product.

## 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\_OK**

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\_OpenResourceForSplF() API.

## **IBM i Access for Windows APIs for new spooled files**

The following IBM i Access for Windows APIs pertain to working with new spooled files. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwbOBJ\_CloseNewSplF:**

Use this API with the IBM i Access for Windows product.

## **Purpose**

Closes a newly created spooled file.

## **Syntax**

```
unsigned int CWB_ENTRY cwbOBJ_CloseNewSplF(  
    cwbOBJ_ObjHandle newSplFHandle,  
    cwbSV_ErrHandle errorHandle);
```

## **Parameters**

### **cwbOBJ\_ObjHandle newSplFHandle - input**

New spooled file handle. This is the handle passed back on the cwbOBJ\_CreateNewSplF() 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\_OK**

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\_CloseNewSplFAndGetHandle:**

Use this API with the IBM i Access for Windows product.

### Purpose

Closes a newly created spooled file and returns a handle to it.

### Syntax

```
unsigned int CWB_ENTRY cwbOBJ_CloseNewSplFAndGetHandle(  
    cwbOBJ_ObjHandle    newSplFHandle,  
    cwbOBJ_ObjHandle    *splFHandle,  
    cwbSV_ErrHandle     errorHandle);
```

### Parameters

#### **cwbOBJ\_ObjHandle newSplFHandle - input**

New spooled file handle. This is the handle passed back on the cwbOBJ\_CreateNewSplF() API.

#### **cwbOBJ\_ObjHandle \*splFHandle - 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\_OK**

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 splFHandle must be released with the cwboBJ\_DeleteObjHandle() API in order to free resources.

## cwboBJ\_CreateNewSplF:

Use this API with the IBM i Access for Windows product.

## Purpose

Creates a new IBM i spooled file.

## Syntax

```
unsigned int CWB_ENTRY cwboBJ_CreateNewSplF(  
    const char *systemName,  
    cwboBJ_ParmHandle *parmListHandle,  
    cwboBJ_ObjHandle *printerFileHandle,  
    cwboBJ_ObjHandle *outputQueueHandle,  
    cwboBJ_ObjHandle *newSplFHandle,  
    cwboSV_ErrHandle errorHandle);
```

## Parameters

### const char \*systemName - input

Pointer to the system name contained in ASCII 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 CWBOBJ\_KEY\_OUTQUELIB & CWBOBJ\_KEY\_OUTQUE) it will override the output queue specified by this output queue handle.

### cwboBJ\_ObjHandle \*newSplFHandle - output

A pointer to a 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.

### cwboSV\_ErrHandle errorHandle - output

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwboSV\_CreateErrHandle() API. The messages may be retrieved through the cwboSV\_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 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/QNPSPRTF. 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_MULTIUPL	- 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_USRDFNDDTA	- 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

**Note:**

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\_GetSpIFHandleFromNewSpIF:**

Use this API with the IBM i Access for Windows product.

**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,
    cwSV_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\_OK**

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\_SPLFNOPEN**

Spooled file hasn't been created on the host yet.

### **Usage**

The handle returned in `splFHandle` 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 wasn't specified on the `cwbOBJ_CreateNewSplF()` 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_SPLFNOPEN`.

### **cwbOBJ\_WriteNewSplF:**

Use this API with the IBM i Access for Windows product.

### **Purpose**

Writes data into a newly created spooled file.

### **Syntax**

```
unsigned int CWB_ENTRY cwbOBJ_WriteNewSplF(  
    cwbOBJ_ObjHandle newSplFHandle,  
    const char *data,  
    unsigned long dataLen,  
    cwbSV_ErrHandle errorHandle);
```

### **Parameters**

#### **cwbOBJ\_ObjHandle newSplFHandle - input**

New spooled file handle. This is the handle passed back on the `cwbOBJ_CreateNewSplF()` 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\_OK**

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

## **APIs for reading spooled files for IBM i Access for Windows**

The following IBM i Access for Windows APIs pertain to reading spooled files. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwbOBJ\_CloseSpIF:**

Use this API with the IBM i Access for Windows product.

### **Purpose**

Closes an IBM i spooled file that was previously opened for reading.

### **Syntax**

```
unsigned int CWB_ENTRY  cwbOBJ_CloseSpIF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbSV_ErrHandle  errorHandle);
```

### **Parameters**

#### **cwbOBJ\_ObjHandle splFHandle - 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\_OK**

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\_OpenSplF:**

Use this API with the IBM i Access for Windows product.

**Purpose**

Opens an IBM i spooled file for reading.

**Syntax**

```

unsigned int CWB_ENTRY  cwbOBJ_OpenSplF(
                        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\_OK**

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\_CloseSplF() API when done reading from it.

**cwbOBJ\_ReadSplF:**

Use this API with the IBM i Access for Windows product.

## Purpose

Reads bytes from the current read location.

## Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_ReadSp1F(  
                        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\_OK**

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_OpenSp1F()` 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\_SeekSplF:**

Use this API with the IBM i Access for Windows product.

### **Purpose**

Moves the current read position on a spooled file that is open for reading.

### **Syntax**

```
unsigned int CWB_ENTRY  cwbOBJ_SeekSplF(  
                        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\_OK**

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_OpenSp1F()` API must be called with this spooled file handle before this API is called.

## APIs for manipulating spooled files for IBM i Access for Windows

The following IBM i Access for Windows APIs pertain to manipulating spooled files. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### **cwbOBJ\_CallExitPgmForSp1F:**

Use this API with the IBM i Access for Windows product.

## Purpose

Instructs the IBM i 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_CallExitPgmForSp1F(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        void               *data,  
                        unsigned long      dataLen,  
                        cwbSV_ErrHandle    errorHandle);
```

## Parameters

### **cwbOBJ\_ObjHandle splFHandle - 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\_OK**

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\_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 IBM i exit programs registered with the QNPSERVR program are 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 IBM i 'Guide to Programming for Print' for information on the interface between the QNPSERVR server program and the exit programs.

## cwbOBJ\_CreateSplFHandle:

Use this API with the IBM i Access for Windows product.

## Purpose

Create a spooled file handle for a particular spooled file on a specified system.

## Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_CreateSplFHandle(  
    const char          *systemName,  
    const char          *jobName,  
    const char          *jobNumber,  
    const char          *jobUser,  
    const char          *splFName,  
    const unsigned long splFNumber,  
    cwbOBJ_ObjHandle   *objectHandle,  
    cwbSV_ErrHandle    errorHandle);
```

## Parameters

### const char \*systemName - input

Pointer to the system name contained in an ASCII string.

### const char \*jobName - input

Pointer to the name of the IBM i job that created the spooled file in an ASCII string.

### const char \*jobNumber - input

Pointer to the number of the IBM i job that created the spooled file in an ASCII string.

### const char \*jobUser - input

Pointer to the user of the IBM i job that created the spooled file in an ASCII string.

### const char \*splFName - input

Pointer to the name of the spooled file in an ASCII string.

### const unsigned long splFNumber - 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.

### **cwOBJ\_CreateSplFHandleEx:**

Use this API with the IBM i Access for Windows product.

## Purpose

Create a spooled file handle for a particular spooled file on a specified system.

## Syntax

```
unsigned int CWB_ENTRY cwOBJ_CreateSplFHandleEx(
    const char          *systemName,
    const char          *jobName,
    const char          *jobNumber,
    const char          *jobUser,
    const char          *splFName,
    const unsigned long splFNumber,
    const char          *createdSystem,
    const char          *createdDate,
    const char          *createdTime,
    cwOBJ_ObjHandle    *objectHandle,
    cwSV_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 IBM i job that created the spooled file in an ASCIIZ string.

### **const char \*jobNumber - input**

Pointer to the number of the IBM i job that created the spooled file in an ASCIIZ string.

### **const char \*jobUser - input**

Pointer to the user of the IBM i job that created the spooled file in an ASCIIZ string.

### **const char \*splFName - input**

Pointer to the name of the spooled file in an ASCIIZ string.

**const unsigned long splFNumber - 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 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\_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\_DeleteSplF:**

Use this API with the IBM i Access for Windows product.

## Purpose

Delete an IBM i spooled file.

## Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_DeleteSplF(  
                        cwbOBJ_ObjHandle splFHandle,  
                        cwbSV_ErrHandle  errorHandler);
```

## 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\_OK**

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\_DisplaySplF:**

Use this API with the IBM i Access for Windows product.

### **Purpose**

Displays the specified spooled file to the user.

### **Syntax**

```
unsigned int CWB_ENTRY  cwbOBJ_DisplaySplF(
                        cwbOBJ_ObjHandle    splFHandle,
                        const char           *view,
                        const unsigned long  flags,
                        cwbSV_ErrHandle     errorHandle);
```

### **Parameters**

#### **cwbOBJ\_ObjHandle splFHandle - input**

Handle of the parameter object.

#### **const char \*view - input**

Optional, may be NULL. If specified it is a pointer to an ASCIIZ 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 following bits may be set: `CWBOBJ_DSPSPLF_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\_HoldSpIF:**

Use this API with the IBM i Access for Windows product.

**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\_OK**

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:**

Use this API with the IBM i Access for Windows product.

### 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:**

Use this API with the IBM i Access for Windows product.

## Purpose

Moves an IBM i spooled file to another output queue or to another position on the same output queue.

## Syntax

```
unsigned int CWB_ENTRY cwboBJ_MoveSpIF(  
    cwboBJ_ObjHandle  splFHandle,  
    cwboBJ_ObjHandle *targetSpIFHandle,  
    cwboBJ_ObjHandle *outputQueueHandle,  
    cwboSV_ErrHandle  errorHandle);
```

## Parameters

### **cwboBJ\_ObjHandle splFHandle - 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.

### **cwboSV\_ErrHandle errorHandle - output**

Optional, may be 0. Any returned messages will be written to this object. It is created with the cwboSV\_CreateErrHandle() API. The messages may be retrieved through the cwboSV\_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 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 `targetSplFHandle` and `outputQueueHandle` are `NULL`, the spooled file will be moved to the first position on the current output queue.

### **cwbOBJ\_ReleaseSplF:**

Use this API with the IBM i Access for Windows product.

## Purpose

Releases a spooled file.

## Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_ReleaseSplF(  
                        cwbOBJ_ObjHandle  splFHandle,  
                        cwbSV_ErrHandle  errorHandler);
```

## Parameters

### **cwbOBJ\_ObjHandle splFHandle - input**

Handle of the spooled file to be released.

### **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\_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\_SendNetSplF:**

Use this API with the IBM i Access for Windows product.

## 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_SendNetSp1F(  
                        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\_OK**

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\_SendTCPSP1F:**

Use this API with the IBM i Access for Windows product.

#### **Purpose**

Sends a spooled file to be printed on a remote system. This is the IBM i version of the TCP/IP LPR command.

#### **Syntax**

```
unsigned int CWB_ENTRY  cwbOBJ_SendTCPSP1F(  
                        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\_OK**

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 IBM i send TCP/IP spooled file (SNDTCPSPLF) command is issued against the spooled file. The following parameter key's MUST be set in the parmListHandle object:

- CWBOBJ\_KEY\_RMTSYSTEM  
Specifies the remote system to which the print request is sent. May be a remote system name or `"*INTERNETADR"`.
- CWBOBJ\_KEY\_RMTTPRTQ  
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 IBM i types, 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\_WSCUSTOMOBJL  
Specifies the name of the workstation customizing object library.

## APIs for handling spooled file messages for IBM i Access for Windows

The following IBM i Access for Windows APIs pertain to handling spooled file messages. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### cwbOBJ\_AnswerSpIFMsg:

Use this API with the IBM i Access for Windows product.

### Purpose

Answer the message that the spooled file is waiting on.

## Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_AnswerSplFMsg(  
                        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\_OK**

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\_GetSplFMsgAttr:**

Use this API with the IBM i Access for Windows product.

## Purpose

Retrieves an attribute of a message that's associated with a spooled file.

## Syntax

```
unsigned int CWB_ENTRY  cwbOBJ_GetSplFMsgAttr(  
                        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\_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\_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.

## APIs for analyzing spooled file data for IBM i Access for Windows

The following IBM i Access for Windows APIs pertain to analyzing spooled file data. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

### cwbOBJ\_AnalyzeSpIFData:

Use this API with the IBM i Access for Windows product.

#### 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_SplIFDataType *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\_SplIFDataType \*dataType - output

On output this will contain the data type. If the data type can not be determined, it defaults to CWBOBJ\_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.

**Server program APIs for IBM i Access for Windows**

The following IBM i Access for Windows APIs pertain to server programs. The APIs are listed alphabetically.

**Note:** When working with handles in the following APIs, 0 never will be returned as a valid handle.

**cwbOBJ\_DropConnections:**

Use this API with the IBM i Access for Windows product.

**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\_OK**

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\_GetNPServerAttr:**

Use this API with the IBM i Access for Windows product.

## Purpose

Get an attribute of the QNPSERVER 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 errorHandle);
```

## 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 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\_BUFFER\_OVERFLOW**

Buffer too small.

### **CWB\_INVALID\_PARAMETER**

Invalid parameter specified.

### **CWBOBJ\_RC\_HOST\_ERROR**

Host error occurred. Text may be in errorHandle.



## **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 QNPSEVR program:

- CWBOBJ\_KEY\_NPSCCSID - Server CCSID
- CWBOBJ\_KEY\_NPSLEVEL - Server code level

### **cwbOBJ\_SetConnectionsToKeep:**

Use this API with the IBM i Access for Windows product.

### **Purpose**

Set the number of connections that should be left active for a particular system. Normally, the cwboj.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.

## Example: Using system objects APIs for IBM i 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 (ulRC == CWB_OK)
            {
                /* walk through the list of items, displaying */
                /* each item to the user */

                ulObjPosition = 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);
}

```

## IBM i Access for Windows Remote Command/Distributed Program Call APIs

The IBM i Access for Windows Remote Command/Distributed Program Call APIs allow the PC application programmer to access IBM i functions. User program and system commands are called without requiring an emulation session. A single IBM i program serves commands and programs, so only one system job is started for both.

### IBM i Access for Windows Remote Command APIs:

The IBM i Access for Windows Remote Command application programming interfaces (APIs) enable your PC application to start non-interactive IBM i commands and to receive completion messages from these commands. The IBM i command can send up to ten reply messages.

### IBM i Access for Windows Distributed Program Call API:

The IBM i Access for Windows Distributed Program Call API allows your PC application to call any IBM i 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 IBM i program that was called. If the program fails to run correctly on the system, the program can send up to ten reply messages.

### IBM i 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**.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

#### Related reference

"Remote Command/Distributed Program Call APIs return codes" on page 28

There are IBM i Access for Windows Remote command and distributed program call API return codes.

"IBM i name formats for connection APIs" on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

"OEM, ANSI, and Unicode considerations" on page 6

Most of the IBM i Access for Windows C/C++ APIs that accept string parameters exist in three forms: OEM, ANSI, or Unicode.

### Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs

An application that uses the IBM i 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 is an IBM i identification. 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 an IBM i request. 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 the IBM i Access for Windows product.

**Program object**

This represents the IBM i 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 function. This function returns a handle to the system object. This handle then is used with the cwbRC\_StartSysEx function to start an IBM i conversation. 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
- cwbRC\_CallPgm
- cwbRC\_RunCmd
- cwbRC\_StopSys

A command is a character string that is to be run on the IBM i platform. 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 IBM i 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\_AddParm
- cwbRC\_CreatePgm
- cwbRC\_DeletePgm
- cwbRC\_GetLibName
- cwbRC\_GetParm
- cwbRC\_GetParmCount
- cwbRC\_GetPgmName
- cwbRC\_SetLibName
- cwbRC\_SetParm
- cwbRC\_SetPgmName

## Related reference

“cwbCO\_CreateSystem” on page 47

Use the IBM i Access for Windows cwbCO\_CreateSystem command.

“cwbRC\_StartSysEx” on page 342

Use this API with the IBM i Access for Windows product.

“cwbRC\_CallPgm” on page 346

Use this API with the IBM i Access for Windows product.

“cwbRC\_RunCmd” on page 344

Use this API with the IBM i Access for Windows product.

“cwbRC\_StopSys” on page 343

Use this API with the IBM i Access for Windows product.

“cwbRC\_CreatePgm” on page 347

Use this API with the IBM i Access for Windows product.

“cwbRC\_AddParm” on page 345

Use this API with the IBM i Access for Windows product.

“cwbRC\_GetParmCount” on page 351

Use this API with the IBM i Access for Windows product.

“cwbRC\_GetParm” on page 350

Use this API with the IBM i Access for Windows product.

“cwbRC\_GetPgmName” on page 352

Use this API with the IBM i Access for Windows product.

“cwbRC\_GetLibName” on page 349

Use this API with the IBM i Access for Windows product.

“cwbRC\_SetParm” on page 354

Use this API with the IBM i Access for Windows product.

“cwbRC\_SetPgmName” on page 355

Use this API with the IBM i Access for Windows product.

“cwbRC\_SetLibName” on page 353

Use this API with the IBM i Access for Windows product.

“cwbRC\_DeletePgm” on page 349

Use this API with the IBM i Access for Windows product.

## Remote Command/Distributed Program Call: Access remote command APIs list for IBM i Access for Windows

Access the IBM i remote command server program. The request handle is used to run commands and to call programs. The APIs are listed alphabetically.

### cwbRC\_GetClientCCSID:

Use this API with the IBM i Access for Windows product.

### 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 IBM i 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 is the IBM i identification.

### **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:**

Use this API with the IBM i Access for Windows product.

## Purpose

Get the coded character set identifier (CCSID) associated with the IBM i job. This CCSID along with the client CCSID can be used to convert EBCDIC data returned by some IBM i 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 is the IBM i identification.

### **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 CWBNLCNV.H file.

### **cwbRC\_StartSysEx:**

Use this API with the IBM i Access for Windows product.

### **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 IBM i conversation is 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**

The IBM i application is not found.

#### **CWB\_HOST\_NOT\_FOUND**

The system is 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.

**Related reference**

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

**cwBRC\_StopSys:**

Use this API with the IBM i Access for Windows product.

**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 is the IBM i identification.

**Return Codes**

The following list shows common return values:

**CWB\_OK**

Successful completion.

**CWBRC\_INVALID\_SYSTEM\_HANDLE**

Invalid system handle.

**Usage**

None

## Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

## Remote Command/Distributed Program Call: Run APIs list for IBM i Access for Windows

Use these APIs to run an IBM i command. The APIs are listed alphabetically.

### **cwbRC\_RunCmd:**

Use this API with the IBM i Access for Windows product.

### **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 is the IBM i identification.

#### **const char \*commandString - input**

Pointer to a string that contains the command that is issued. This is an ASCII string.

#### **cwbSV\_ErrHandle msgHandle - output**

Any IBM i returned messages are 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

**Related reference**

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

**Remote Command/Distributed Program Call: Access programs APIs list for IBM i Access for Windows**

Use these IBM i Access for Windows APIs to access programs and their parameters.

**cwbRC\_AddParm:**

Use this API with the IBM i Access for Windows product.

**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,  
CWBRC_INOUT | CWBRC_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.

**Related reference**

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

**cwbRC\_CallPgm:**

Use this API with the IBM i Access for Windows product.

**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 is the IBM i identification.

### **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

## Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

### **cwbRC\_CreatePgm:**

Use this API with the IBM i Access for Windows product.

## 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 ASCIIZ string that contains the name of the program that you want to call. The name is uppercased unless enclosed in double quotes.

### **const char \*libraryName - input**

Pointer to an ASCIIZ string that contains the name of the library where the program resides. The name is uppercased unless enclosed in double quotes.

### **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 IBM i program object for each program you want to call on the system. 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.

## Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

## **cwbRC\_DeletePgm:**

Use this API with the IBM i Access for Windows product.

## 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.

## Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

## **cwbRC\_GetLibName:**

Use this API with the IBM i Access for Windows product.

## 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

## Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

## **cwbRC\_GetParm:**

Use this API with the IBM i Access for Windows product.

## 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

**Related reference**

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

**cwbRC\_GetParmCount:**

Use this API with the IBM i Access for Windows product.

**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

## Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

### **cwbRC\_GetPgmName:**

Use this API with the IBM i Access for Windows product.

## 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

**Related reference**

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

**cwbRC\_SetLibName:**

Use this API with the IBM i Access for Windows product.

**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.

### Related reference

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

### cwbRC\_SetParm:

Use this API with the IBM i Access for Windows product.

### 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_SetParm( 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.

**Related reference**

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

**cwBRC\_SetPgmName:**

Use this API with the IBM i Access for Windows product.

**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.

### **Related reference**

“Typical use of IBM i Access for Windows Remote Command/Distributed Program Call APIs” on page 338

An application that uses the IBM i Access for Windows Remote Command/Distributed Program Call function uses objects.

## **Example: Using Remote IBM i Access for Windows Command/Distributed Program Call APIs**

This example illustrates using remote IBM i 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>  
using namespace std;  
#include <TCHAR.H>
```

```

#include "cwbrc.h"
#include "cwbcosys.h"
/*****

void main()
{
    cwbcO_SysHandle system;
    cwbcRC_SysHandle request;
    cwbcRC_PgmHandle program;

    // Create the system object
    if ( (cwbcO_CreateSystem("SystemName",&system)) != CWB_OK )
        return;

    // Start the system
    if ( (cwbcRC_StartSysEx(system,&request)) != CWB_OK )
        return;

    // Call the command to create a library
    char* cmd1 = "CRTLIB LIB(RCTESTLIB) TEXT('RC TEST LIBRARY)";
    if ( (cwbcRC_RunCmd(request, cmd1, 0)) != CWB_OK )
        return;

    cout << "Created Library" << endl;

    // Call the command to delete a library
    char* cmd2 = "DLTLIB LIB(RCTESTLIB)";
    if ( (cwbcRC_RunCmd(request, cmd2, 0)) != CWB_OK )
        return;

    cout << "Deleted Library" << endl;

    // Create a program object to create a user space
    if ( cwbcRC_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};

    if ( cwbcRC_AddParm(program, CWBRC_INPUT, 20, name) != CWB_OK)
        return;

    if ( cwbcRC_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;
}

```

## IBM i Access for Windows Serviceability APIs

The IBM i 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 IBM i 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 IBM i Access for Windows Serviceability APIs:

The IBM i 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.

## IBM i 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**.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

### Related reference

"Serviceability APIs return codes" on page 30

There are IBM i Access for Windows serviceability API return codes.

## History log and trace files

History logs and trace files allow you to log information about your IBM i Access for Windows programs.

### History log:

The log functions allow you to write message text to the IBM i Access for Windows History Log. The message text needs to be displayable ASCII character data.

All IBM i Access for Windows programs log messages to the IBM i 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 can 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. IBM i 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 record of flow of control through the code.

IBM i 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.

IBM i 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 IBM i Access for Windows error handle functions allow you to create an error handle (`cwbsV_CreateErrHandle`) to use on IBM i Access for Windows APIs that support it.

If an error occurs (a non-zero return code) on the IBM i 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_GetErrTextIndexed`) for each of the error messages

## **Typical use of Serviceability APIs**

Typical uses of IBM i Access for Windows serviceability APIs include history logs and error handles.

### **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 IBM i 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 system, 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 IBM i 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 IBM i Access return codes.

### **Serviceability APIs list: Writing to history log**

Use these IBM i Access for Windows APIs to write message text to a history log

#### **cwbsv\_CreateMessageTextHandle:**

Use this API with the IBM i Access for Windows product.

#### **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\_DeleteMessageTextHandle:**

Use this API with the IBM i Access for Windows product.

**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\_LogMessageText:**

Use this API with the IBM i Access for Windows product.

**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\_SetMessageClass:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API to set a IBM i Access for Windows message handle.

## 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:**

Use this API to set the IBM i Access for Windows product identifier.

### 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.

## Serviceability APIs list: Writing trace data

Use these IBM i Access for Windows APIs to write trace data to a detail trace file

### **cwbSV\_CreateTraceDataHandle:**

Use this API with the IBM i Access for Windows product.

### 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\_DeleteTraceDataHandle:**

Use this API with the IBM i Access for Windows product.

## 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\_LogTraceData:**

Use this API with the IBM i Access for Windows product.

**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\_SetTraceComponent:**

Use this API with the IBM i Access for Windows trace entries.

## 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:**

Use this API with the IBM i Access for Windows trace entries.

## 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.

**Serviceability API list: Writing trace points**

Use these IBM i Access for Windows APIs to write trace points to an entry/exit trace file

**cwbSV\_CreateTraceAPIHandle:**

Use this API with the IBM i Access for Windows product.

**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\_CreateTraceSPIHandle:**

Use this API with the IBM i Access for Windows product.

## 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\_DeleteTraceAPIHandle:**

Use this API with the IBM i Access for Windows product.

**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\_DeleteTraceSPIHandle:**

Use this API with the IBM i Access for Windows product.

**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\_LogAPIEntry:**

Use this API with the IBM i Access for Windows product.

## 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-byte 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 to 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:

Use this API with the IBM i Access for Windows product.

## 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-byte 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 to 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\_LogSPIEntry:

Use this API with the IBM i Access for Windows product.

## 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 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-byte 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 to 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:**

Use this API with the IBM i Access for Windows product.

#### **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-byte 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\_SetAPIComponent:**

Use this API with the IBM i Access for Windows trace entries.

### 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 features that are defined under it.

#### **cwbSV\_SetAPIProduct:**

Use this API with the IBM i Access for Windows product.

### 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:**

Use this API in setting an IBM i Access for Windows trace entry.

## 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:

Use this API with the IBM i Access for Windows trace entries.

### 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.

### Serviceability API list: Reading service files

Use these IBM i Access for Windows APIs to read service files, service file records, and service file header information. Additionally, you can read history log service records, detail trace file service records, and entry/exit trace file service records.

### cwbSV\_ClearServiceFile:

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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\_CreateServiceRecHandle:**

Use this API with the IBM i Access for Windows product.

## 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\_DeleteServiceRecHandle:**

Use this API with the IBM i Access for Windows product.

## 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\_GetComponent:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

**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\_GetMaxRecordSize:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

### **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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

## 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 timeStringLength,  
    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 timeStringLength - 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\_GetTraceData:**

Use this API with the IBM i Access for Windows product.

**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\_GetTraceAPIData:**

Use this API with the IBM i Access for Windows product.

### **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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

## 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 errorHandler);
```

## 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 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\_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\_GetTraceSPIData:**

Use this API with the IBM i Access for Windows product.

## 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
unsigned long
unsigned long
        *spiData,
        spiDataLength,
        *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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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\_OpenServiceFile:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

## **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:**

Use this API with the IBM i Access for Windows product.

### **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:**

Use this API with the IBM i Access for Windows product.

### **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_Read01destRecord(  
    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:**

Use this API with the IBM i Access for Windows product.

## 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 errorHandle);
```



## 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 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, "`ReadNewestRecord()`" is performed.

## Serviceability API list: Retrieving message text

Use these IBM i Access for Windows APIs to retrieve message text associated with error handles.

### **cwbSV\_CreateErrHandle:**

Use this API with the IBM i Access for Windows product.

## Purpose

This function creates an error message object and returns a handle to it. This error handle can be passed to IBM i 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\_DeleteErrHandle:**

Use this API with the IBM i Access for Windows product.

**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\_GetErrClass:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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 IBM i messages. The file name is the name of the IBM i 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

IBM i messages are sometimes 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 IBM i 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:**

Use this API with the IBM i Access for Windows product.

## Purpose

Returns the message file name for the message identified by the index provided. This message attribute only pertains to IBM i returned messages. The file name is the name of the IBM i 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

IBM i messages are sometimes 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 IBM i 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`:

Use this API with the IBM i Access for Windows product.

## 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 IBM i returned messages. The library name is the name of the IBM i 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**

IBM i messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` APIs. In these cases, you can use this API to retrieve the message file library name for the IBM i 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:**

Use this API with the IBM i Access for Windows product.

**Purpose**

Returns the message file library name for the message identified by the index provided. This message attribute only pertains to IBM i returned messages. The library name is the name of the IBM i 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 ASCIIZ 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**

IBM i messages are sometimes 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 IBM i 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:**

Use this API with the IBM i Access for Windows product.

**Purpose**

Returns the message substitution data for the top-level (the most recent) message identified by the error handle provided. This message attribute only pertains to IBM i returned messages. The substitution data are inserted into the substitution variable fields defined for the message.

**Syntax**

```
unsigned int CWB_ENTRY cwbSV_GetErrSubstText(  
    cwbSV_ErrHandle  errorHandle,  
    char             *substitutionData,  
    unsigned long    substitutionDataLength,  
    unsigned long    *returnLength);
```

## Parameters

### **cwbSV\_ErrHandle errorHandler** - input

Handle that was returned by a previous call to the `cwbSV_CreateErrHandle()` API.

### **char \* substitutionData** - input/output

Pointer to a buffer that will receive the substitution data 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 data are returned as EBCDIC values.

### **unsigned long substitutionDataLength** - 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

IBM i 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 data for the IBM i messages contained in the error handle. If there is no substitution data 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 data when the return code is `CWB_OK`. The substitution data returned on this API could be used on a subsequent host retrieve message API call (`QSYS/QMHRTVM`) to retrieve the format of the substitution data or to return secondary help text with the substitution data added in. Host API's are called using the `cwbRC_CallPgm()` API.

### **cwbSV\_GetErrSubstTextIndexed:**

Use this API with the IBM i Access for Windows product.

## Purpose

Returns the message substitution data for the message identified by the index provided. This message attribute only pertains to IBM i returned messages. The substitution data 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             *substitutionData,  
    unsigned long    substitutionDataLength,  
    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 data 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 \* substitutionData - input/output**

Pointer to a buffer that will receive the substitution data 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 data are returned as EBCDIC values.

### **unsigned long substitutionDataLength - 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

IBM i messages may be added to the error handle when using the `cwbRC_CallPgm()` and `cwbRC_RunCmd()` APIs. In these cases, you can use this API to retrieve the substitution data for the IBM i messages contained in the error handle. If there is no substitution data 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 data when the return code is CWB\_OK. The substitution data returned on this API could be used on a subsequent host retrieve message API call (QSYS/QMHRTVM) to retrieve the format of the substitution data or to return secondary help text with the substitution data added in. Host API's are called using the cwbRC\_CallPgm() API.

### **cwbSV\_GetErrText:**

Use this API with the IBM i Access for Windows product.

### **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:

Use this API with the IBM i Access for Windows product.

### 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.

## Example: Using IBM i Access for Windows Serviceability APIs

The following example uses the IBM i Access for Windows Serviceability APIs to log a message string to the IBM i Access for Windows History Log. Before running this program, start the IBM i Access for Windows Diagnostics History log.

```
#include <stdio.h>
#include <string.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,
    errorHandle)) != 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);
}

```

## IBM i Access for Windows System Object Access (SOA) APIs

System Object Access enables you to view and manipulate system objects through a graphical user interface.

System Object Access application programming interfaces (APIs) for IBM i 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 IBM i Access for Windows required files:

Interface definition file	Import library	Dynamic Link Library
cwboapi.h	cwbapi.lib	cwboapi.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 **IBM i Operations** → **C/C++ APIs**.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

#### Related reference

"System Object Access APIs return codes" on page 31

There are IBM i Access for Windows SOA API return codes.

"IBM i name formats for connection APIs" on page 5

APIs that take an IBM i name as a parameter, accept the name in the three different formats.

### SOA objects

Use System Object Access to view and to manipulate the following IBM i 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

### System object views

Two types of system object views are provided with IBM i Access for Windows.

#### List view:

Displays a customizable graphical list view of the selected system 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 system 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 IBM i Access for Windows

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.

### Display a customized list of system objects:

In this example, a list of IBM i spool file objects 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.

### Display a customized list of system objects (summary)

(O)	cwBRC_StartSys	Start an IBM i conversation
(R)	CWBSO_CreateListHandle	Create a list of system 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 IBM i conversation

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

*Sample program: Display a customized list of system objects:*

Use this sample program to display IBM i objects.

```
#ifdef UNICODE
#define _UNICODE
#endif
#include <windows.h> // Windows APIs and datatypes
#include "cwsoapi.h" // System Object Access APIs
#include "cwbrc.h" // IBM i DPC APIs
#include "cwbn.h" // IBM i Navigator APIs
```

```

#define APP_PROFILE "APPPROF" // 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
cwbCO_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 IBM i 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;
}

```

### Display the Properties view for a system object:

A list object for a list of IBM i 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.

### Display the properties view for an object (Summary)

(O)	cwbRC_StartSys	Start an IBM i conversation
(R)	CWBSO_CreateListHandle	Create a list of system objects
(O)	CWBSO_SetListFilter	Set list filter criteria
(R)	CWBSO_CreateErrorHandle	Create an error object
(R)	CWBSO_OpenList	Open the list (builds an IBM i list)
(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 IBM i conversation

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

*Sample program: Display the Properties view of an object:*

Use this sample program with IBM i Access for Windows to display property views.

```

#ifdef UNICODE
    #define _UNICODE
#endif
#include <windows.h>           // Windows APIs and datatypes
#include "cwsoapi.h"         // System Object Access APIs
#include "cwbrc.h"           // IBM i DPC APIs
#include "cwun.h"            // IBM i 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 IBM i 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;
}

```

### Access and update data for system objects:

A list object for IBM i 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.

In this example, 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.

### Access and update data for system objects (Summary)

```
(R) CWBSO_CreateListHandle      Create a list of system objects
(O) CWBSO_SetListFilter         Set list filter criteria
(R) CWBSO_CreateErrorHandle     Create an error object
(R) CWBSO_OpenList              Open the list (automatically starts
                                an IBM i conversation)
(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 IBM i attribute
.
. (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
                                IBM i conversation)
```

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

*Sample program: Access and update data for system objects:*

Use this IBM i Access for Windows sample program that updates system objects.

```
#include <windows.h>           // Windows APIs and datatypes
#include <stdlib.h>            // For atoi
#include "cwboapi.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
wsprintf(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;
}

```

## IBM i Access for Windows System Object Access programming considerations

See the following topics for important SOA programming considerations.

### About System Object Access errors:



IBM i Access for Windows functions support all System Object Access APIs which 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 IBM i operating system, 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 IBM i Access for Windows install directory. This file is English only and is intended for use by IBM personnel for problem analysis.

#### **Related reference**

“System Object Access APIs return codes” on page 31  
There are IBM i Access for Windows SOA API return codes.

#### **System Object Access application profiles:**

Use IBM i Access for Windows application profiles.

By default, user-specified list filter criteria are not saved to disk. System Object Access provides APIs for the following.

- 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 IBM i 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.

#### **Manage IBM i communications sessions for application programs:**

System Object Access APIs for IBM i Access for Windows communicate with the system 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 IBM i object that is identified on the `CWBSO_CreateListHandleEx` API, a conversation automatically will be started when the list is opened or displayed. If IBM i 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 IBM i conversation is automatically ended (Note that `CWBSO_CloseList` does not end the IBM i conversation).

A System Object Access conversation may be started using the `cwbRC_StartSysEx` API. This API accepts an IBM i 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 IBM i conversation.

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 IBM i Access for Windows List

The following System Object Access APIs for IBM i Access for Windows are listed alphabetically.

### SOA enablers:

System Object Access also includes enablers (APIs), which applications can use to access data in system 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 IBM i list associated with client
             list
(Manipulate the list and its objects using various generic
 and subclass APIs)
CloseList -- Closes list and release IBM i resource
DeleteListHandle -- Destroys list object on the client
```

The `CWBSO_CreateListHandle` 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` 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` 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` API must be called to build the list of objects. This results in a request that is sent to the system. The list is built on the system, and some or all of the objects (records) in the list are buffered down to the list on the client. Although all objects in the list are not necessarily cached on the client, the APIs behave as if they are. Once the `CWBSO_OpenList` API is called successfully, the following APIs can be called:

#### **CWBSO\_GetObjHandle**

Retrieves a handle to a specific object in the list. The object handle can then be used to manipulate the specific object.

#### **CWBSO\_DeleteObjHandle**

Releases the handle returned by `CWBSO_GetObjHandle`.

#### **CWBSO\_DisplayList**

Displays the spreadsheet view of the list.

#### **CWBSO\_GetListSize**

Retrieves the number of objects in the list.

#### **CWBSO\_CloseList**

Closes the IBM i list and destroys 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 API is called again. The CWBSO\_DeleteListHandle API should be called to destroy the list object.

### **CWBSO\_CloseList:**

Use this API with the IBM i Access for Windows product.

#### **Purpose**

Closes the list of objects and frees up IBM i allocated resources.

#### **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 does not end the IBM i conversation. To end the conversation, the list must be deleted using CWBSO\_DeleteListHandle.

### **CWBSO\_CopyObjHandle:**

Use this API with the IBM i Access for Windows product.

## Purpose

Creates a new instance of an object and returns a handle to the new instance. This does not create a new system object. It merely creates an additional instance of a system 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 is set to a new handle for the same system 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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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 IBM i name on which the list is built. The name that is specified must be a configured system. If the client is not currently connected to the system, an IBM i connection is established when the list is opened. If NULL is specified for the system name, the current IBM i Access default system is 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\* 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 IBM i 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:**

Use this API with the IBM i Access for Windows product.

## 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 system on which the list will be built. This IBM i handle must be for a configured system.

### **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 IBM i 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:**

Use this API with the IBM i Access for Windows product.

## 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 system on which the object is built. The name that is specified must be a configured system. If the client is not currently connected, an IBM i connection is established when the list is opened. If NULL is specified for the system name, the current IBM i default system is 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 IBM i 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:**

Use this API with the IBM i Access for Windows product.

### **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:**

Use this API with the IBM i Access for Windows product.

### **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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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:

Use this API with the IBM i Access for Windows product.

## 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:

- cwbsobj.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:

Use this API with the IBM i Access for Windows product.

## 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:

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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 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.

### **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 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 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\_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. 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:**

Use this API with the IBM i Access for Windows product.

**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:**

Use this API with the IBM i Access for Windows product.



## 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 IBM i errors, the message text is in the language that is specified for the user's execution environment. All other message text are in the language that is specified in the Windows Control Panel on the user's personal computer.

### **CWBSO\_GetListSize:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

#### **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
- cwbsoprt.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 454 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:**

Use this API with the IBM i Access for Windows product.

## 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 is set to the handle for the IBM i 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:**

Use this API with the IBM i Access for Windows product.

**Purpose**

Opens the list. A request is sent to the 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 IBM i Access for Windows allocated resources.
- Call `CWBSO_DeleteListHandle` to free up resources that are allocated on the client.

### **CWBSO\_ReadListProfile:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## Purpose

Refreshes an object's IBM i attributes. 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:**

Use this API with the IBM i Access for Windows product.

## 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:**

Use this API with the IBM i Access for Windows product.

## 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 system 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 454 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:**

Use this API with the IBM i Access for Windows product.

**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:

Use this API with the IBM i Access for Windows product.

### 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:

- cwbsojob.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:**

Use this API with the IBM i Access for Windows product.

#### **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:**

Use this API with the IBM i Access for Windows product.

#### **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:**

Use this API with the IBM i Access for Windows product.

## 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`
- `cwbsoprnt.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 454 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 IBM i object's attributes. You must call `CWBSO_SetObjAttr` to actually update the IBM i attribute value or values for the specified object.

### **CWBSO\_WaitForObj:**

Use this API with the IBM i Access for Windows product.

### **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:**

Use this API with the IBM i Access for Windows product.

## 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 saves any filter criteria that was changed by the user during the `CWBSO_DisplayList` API. Filter information is saved in the registry by the system and by type of list. For example, if your application accesses objects from two different 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 IBM i Access for Windows topics that are listed below provide a description of special values that are returned by `CWBSO_GetObjAttr`, and specified on `CWBSO_SetObjAttr`, for each type of object. In addition, any special values that are specified on `CWBSO_SetListFilter` for each type of list object are discussed.



**Special considerations:**

- For attributes that are numeric, it is common practice for IBM i 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 IBM i Application programming interfaces topic in the IBM i Information Center.

*Job attributes:*

System Object Access uses the List Job (QUSLJOB) and Retrieve Job Information (QUSRJOBI) IBM i APIs to retrieve attributes for jobs.

The possible special values are the same as those that are documented in the IBM i APIs: Work Management APIs topic in the IBM i 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) IBM i API to retrieve attributes for messages.

The possible special values are the same as those that are documented in the IBM i APIs: Message Handling APIs topic in the IBM i 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 IBM i 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.

#### **\*DEV**

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 and the 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. Supply a special value of *"\*ALL"* to request a list of all IBM i printers.

*Printer output attributes:*

System Object Access uses the List Spooled Files (QUSLSPL) and Retrieve Spooled File Attributes (QUSRSPLA) IBM i APIs to retrieve attributes for printer output.

The possible special values are the same as those that are documented in the IBM i APIs: Spooled File APIs topic in the IBM i 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 IBM i API List Network Interfaces (QtocLstNetIfc) to retrieve attributes for TCP/IP interfaces.

To retrieve attributes used by System Object Access for TCP/IP interfaces use one of the following APIs.

- Change IPv4 Interface (QTOCC4IF) API
  - This API is documented by a program temporary fix (PTF). For the PTF details, enter SI17284 in the search function on the following page:
    - IBM i Access for Windows Service Packs (<http://www.ibm.com/servers/eserver/iserries/access/casp.htm>)
- List Network Interfaces (QtocLstNetIfc) API

*Ethernet lines attributes:*

You can find information about Ethernet lines in the Configuration IBM i Access for Windows APIs topic.



See the General Configuration APIs topic in the IBM i Information Center.

*Token-ring lines attributes:*

You can find information about token-ring lines in the Configuration IBM i Access for Windows APIs topic.

See the General Configuration APIs topic in the IBM i Information Center.

*Hardware resources attributes:*

You can find information about hardware resources in the Hardware Resource IBM i Access for Windows APIs topic.

See the Hardware Resource APIs topic in the IBM i Information Center.

*Software products attributes:*

You can find information about software products in the Software Product IBM i Access for Windows APIs topic.

See the Software Product APIs topic in the IBM i Information Center.

*TCP/IP routes attributes:*

System Object Access uses the IBM i 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=Minmum 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:*

Use this list to identify valid IBM i users and groups special values.

- 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 IBM i platform.

- 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:*

You can find information about libraries in QSYS in the IBM i Access for Windows Object APIs topic.

See the Object APIs topic in the IBM i Information Center.

---

## IBM i Access for Windows: Database programming

There are multiple IBM i Access for Windows programming interfaces for accessing database files.

Some of the common interfaces allow you to write a single application to access both IBM i and non-IBM i databases. You can use Structured Query Language (SQL) to access DB2<sup>®</sup> for i database files. You can also use stored procedures and record-level access interfaces for access to single records within a file.

The topics below provide information on the interfaces that are supported. Also, see the DB2 for i SQL Reference topic collection in the IBM i Information Center to access the DB2 for i SQL Programming book for additional details.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

### Related information

DB2 for i SQL Reference

## IBM i Access for Windows .NET provider

IBM i Access for Windows .NET provider allows .NET managed programs to access the IBM i database files using SQL.


Your IBM i Access for Windows .NET support is known by any of the following:

- **Managed Provider**
- **DB2 for IBM i .NET Provider**
- **IBM.Data.DB2.iSeries data provider**

Regardless of the name that is referenced, this data provider allows development and support for your PC-to-IBM i SQL applications, when the .NET Data Access Framework on your IBM i connection. It consists of a set of classes and data types that provide access to connection, command, DataAdapter, and DataReader functions as defined and supported by the ADO.NET architectural model.

The **IBM.Data.DB2.iSeries data provider** complements the existing OLE DB database providers. It allows you to use Visual Basic and C# to develop your .NET client/server applications. You can use the Programmer’s Toolkit along with this provider to make development of your .NET Windows client PC applications quicker and easier.


The **Managed Provider** follows the .NET Framework specifications for managed code, including the requirement to have the .NET Framework already installed on your PC. Once the framework is installed, see the User’s Guide for information on installing or removing an IBM i Access for Windows feature.

See Microsoft Web site  for the architecture and details on Microsoft’s .NET Framework, ADO.NET, Windows Installer, GAC, the CLR, and specifications for managed code.

### To access technical details:

- The **DB2 for IBM i .NET Provider Technical Reference**, which is shipped with the IBM i Access for Windows product, provides complete documentation of the **Managed Provider’s** support. To access this information, use this path: **Start** → **Programs** → **IBM i Access for Windows** → **Programmer’s Toolkit** → **.NET Provider Technical Reference**.
- Technical information about the .NET Provider is also available in Visual Studio 2005 and 2008, by filtering on “IBM i Access for Windows”

## .NET framework

See Microsoft Web site  for the architecture and details on Microsoft's .NET Framework, ADO.NET, Windows Installer, GAC, the CLR, and specifications for managed code.

### To install Programmer's Toolkit :

- You can optionally install the Programmer's Toolkit when you install the IBM i Access for Windows product or you can run a **modified setup** after the product is already installed. See Programmer's Toolkit.

### Other .NET information resources:

- IBM i Access for Windows .NET Provider Web site 
- IBM Redbook Integrating DB2 Universal Database™ for iSeries® with Microsoft ADO .NET. SG24-6440 

### iDB2CommandBuilder restrictions on pre-V5R2M0 servers

Due to IBM i limitations i5/OS® releases prior to V5R2M0, using the iDB2CommandBuilder, on these systems has limited support.

Properly specifying Select command text on the iDB2Command object used with the iDB2CommandBuilder is key when connecting to pre-V5R2M0 servers. Here are some recommended guidelines for creating Select statements for use on pre-V5R2M0 servers.

- Simple statements produce the best results. For example, `SELECT * FROM MYSCHEMA.MYTABLE.`
- Fully qualify the table name with its schema. For example, `MYSCHEMA.MYTABLE.`
- Selection fields are allowed, but must be specified in simple format. Only columns specified in the query table should be used. For example, `SELECT ID, NAME, BALANCE FROM MYSCHEMA.MYTABLE.`
- Derived fields or constants in the selection criteria are discouraged. They may produce unpredictable results. For example, `SELECT ID, LENGTH(NAME), 'Name' FROM MYSCHEMA.MYTABLE.`

## IBM i Access for Windows OLE DB provider

Supports record-level access and SQL access to IBM i database files. Use the ActiveX Data Objects (ADO) and the OLE DB interfaces to take advantage of this support.

The IBM i Access for Windows OLE DB Providers, along with the Programmer's Toolkit, make IBM i client/server application development quick and easy from the Windows client PC. The IBM i Access for Windows OLE DB Provider component gives IBM i programmers record-level access interfaces to IBM i logical and physical DB2 for i database files. In addition, they provide support for SQL, data queues, programs, and commands.

ADO and OLE DB standards provide programmers with consistent interfaces to IBM i data and services. All three of the providers (the **IBMDA400**, the **IBMDASQL**, and the **IBMDARLA**) handle all IBM i-to-PC and data type-to-data type conversions.

### To install OLE DB Provider:

See the topics in the User's Guide on installing and removing features to install this provider.

**Note:** The OLE DB Provider is not installed if the computer does not have MDAC 2.5 or later installed, before installing the IBM i Access for Windows product. MDAC can be downloaded from the Microsoft Web site: [www.microsoft.com/data/doc.htm](http://www.microsoft.com/data/doc.htm).

### To access OLE DB Technical Reference:



The IBM i Access for Windows OLE DB Technical Reference, which is shipped with the IBM i

Access for Windows product, provides complete documentation of OLE DB Provider support. To access this information, select **Start** → **Programs** → **IBM i Access for Windows** → **Programmer's Toolkit** → **OLE DB Provider Technical Reference**.

### To install Programmer's Toolkit:

See the topics in the User's Guide on installing and removing features to install this toolkit.

### Other OLE DB information resources:

- IBM i Access for Windows OLE DB Support Web site. 
- IBM Redbook A Fast Path to AS/400® Client/Server Using AS/400 OLE DB Support: SG24-5183 

### Related reference

"ActiveX programming" on page 577

ActiveX automation is a programming technology that is defined by Microsoft and is supported by the IBM i Access for Windows product.

## IBM i Access ODBC

ODBC is a common database interface that uses SQL as its database access language. An ODBC driver is supported by IBM i Access products to provide support for this interface.

### 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 recompiling.
- 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) and also in query tools (where the select statement is created at run time).

## Structured Query Language (SQL):

SOL is a standardized language for defining and manipulating data in a relational database. In accordance with the relational model of data, the database is perceived as a set of tables, relationships are represented as values in tables, and data is retrieved by specifying a result table that can be derived from



one or more base tables. The ODBC API uses dynamic SQL to interact with the database. Dynamic SQL allows the SQL statements to be constructed and executed when the ODBC application is executed.

For more information on SQL, see the DB2 for IBM i *SQL Reference* book. View an HTML online version of the book, or print a PDF version, from the DB2 for IBM i SQL Reference topic collection, in the IBM i Information Center. See the related links below.

### **IBM i Access ODBC topics:**


**Note:** The information linked to from this page applies to the IBM i Access for Windows 32-bit ODBC driver support, the IBM i Access for Windows 64-bit ODBC driver support, and the IBM i Access for Linux® ODBC driver support. For additional information regarding setup in the IBM i Access for Linux environment, choose the link provided below to the IBM i Access for Linux topic collection, in the IBM i Information Center.

You can find documentation on the ODBC standard by searching for ODBC at the Microsoft Web site.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

#### **Related information**

DB2 for i SQL Reference

 System i Access for Windows Linux

 Microsoft Web site

### **Files required to build an ODBC application**

Identify the IBM i Access for Windows files required to build an ODBC application.

Choose from the following topics for information on files and other concepts used in building an ODBC application.

**Note:** 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**.

#### **Choose an interface to access the ODBC driver:**

There are different programming interfaces that can be used with the IBM i Access ODBC Driver. Each interface has its strengths and weaknesses.

Some of the more common programming interfaces are ActiveX Data Objects (ADO), ADO.NET, Rapid Application Development (RAD) tools, and ODBC APIs. The supported languages, reasons for using, and sources of more information for these 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: [www.microsoft.com/data/doc.htm](http://www.microsoft.com/data/doc.htm)
  - More on using the IBM i Access OLE-DB Provider through ADO refer to: “IBM i Access for Windows OLE DB provider” on page 469
- 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;"
```

## ADO.NET

One of the newer programming technologies is Microsoft’s .NET Framework. The ODBC driver can be used from a .NET application by using Microsoft’s System.Data.Odbc .NET provider.

## 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: [www.microsoft.com/data/doc.htm](http://www.microsoft.com/data/doc.htm).
  - For more information about driver-specific features see “Implementation issues of ODBC APIs” on page 490

## ODBC C/C++ application header files:

Identify C/C++ header files and libraries used in a ODBC C/C++ application. These files are supplied by Microsoft and are not shipped as part of IBM i Access for Windows

Header files	Import library	Dynamic Link Library
sql.h	odbc32.lib	odbc32.dll
sqlext.h		
sqltypes.h		
sqlucode.h		

### ODBC APIs: General concepts:

The following general concepts apply to IBM i Access 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 four 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 as 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, statement, or descriptor). 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.

#### 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 using IBM i Access ODBC.

When you use **SQLPrepare**, the statement that contains the parameter markers is passed to the data source to be prepared by the DB2 for i “Optimizer” on page 542. 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 when using these commands with IBM i Access for Windows functions. 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 application 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.

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.

### **Code 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 IBM i Access 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

## Related reference

“Example: Visual Basic - Access and return data by a call to a procedure” on page 573

A Visual Basic example demonstrates creating, preparing, binding, and calling a DB2 for IBM i procedure.

*Call stored procedures:*

Use stored procedures to improve the performance and function of an IBM i Access ODBC application.

Any IBM i program can act as a stored procedure. i 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 the Stored procedures topic for more information.

To call a stored procedure, complete the following steps:

1. Verify that the stored procedure has been defined by using the 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. For more information on CREATE PROCEDURE and DROP PROCEDURE statements, refer to the DB2 for i SQL Reference topic in the IBM i Information Center.

2. Prepare the call of the stored procedure by using *SQL Prepare*.
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 external procedure named NEWORD, written in the COBOL language, and located  
| in the SQL PATH, 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);
```

## Related reference

“SQL and External procedures” on page 562

SQL and external procedures are supported on IBM i for database access.

## Related information

DB2 for i SQL Reference

*Multiple row INSERT and multiple row FETCH examples:*

Multiple row inserts and multiple row fetches can be used to enhance the performance of an IBM i Access ODBC application.

They allow you to insert or retrieve in multiple rows, rather than individual rows. This reduces the data flows and line turnaround between the client and the server. Multiple row fetches can be accomplished using either the SQLFetch (forward only) or SQLExtendedFetch or SQLFetchScroll API.

A multiple row fetch:

- Returns multiple rows 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 multiple row insert of 6 rows of data followed by two multiple row 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 multiple row 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 multiple row insert
rc = SQLExecDirect(hstmt, "INSERT INTO ITEMS ? ROWS VALUES(?,?)", SQL_NTS);

// set up things for the multiple row 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 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...
// ...

```

### Related reference

“ODBC support for multiple row statements” on page 544

DB2 for IBM i and ODBC supports multiple row operations on INSERT, UPDATE, DELETE, and MERGE statements using the technique described below. This example shows how to use the multiple row INSERT statement in ODBC to insert multiple rows into a DB2 for i table.

*Example: Multiple row inserts using Visual Basic:*

This example is an IBM i Access for Windows Visual Basic multiple row insert that is significantly faster than a “parameterized” insert.

**Multiple row inserts** allow you to:

- Insert blocks of records with one SQL call.
- Reduces the flows between the client and server.

See “Multiple row INSERT and multiple row FETCH examples” on page 476 for additional information.

```

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 IBM i 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, IBM i 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

**Retrieve results:**

When using IBM i Access for Windows functions, in order to work with the rows in a result set, call one of the ODBC APIs which retrieves data. These APIs are the **SQLFetch** , **SQLExtendedFetch**, and **SQLFetchScroll** APIs. These APIs can be used to retrieve one or more rows of the result set.

By running queries or CALL statements SQL can returns result sets to the application program. Running an SQL SELECT statement returns the selected rows in a result set. The **SQLFetch** API can then be used to sequentially retrieve the rows from the result set into the application program's internal storage.

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 in the table. You may not know what columns or how many columns will be returned. You can use the **SQLNumResultCols** API to retrieve the number of result columns in the result set. The **SQLDescribeCol** API can be used to obtain a description of the attributes of each column in the result set.

### **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

```

### Access a database server with an ODBC application:

An IBM i Access ODBC application needs to follow a basic set of steps in order to access a database.

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.

### Establish ODBC connections:

Use these handle types to establish an IBM i Access ODBC connection.

#### SQLAllocHandle with SQL\_HANDLE\_ENV as the handle type

- Allocates memory for an environment handle.
  - Identifies storage for global information:
    - Valid 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).

### Execute ODBC functions:

Use these handle types to execute IBM i Access 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

## Execute prepared statements:

If a SQL IBM i Access ODBC 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 across 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 IBM i 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. See the topic below on the performance architecture of the driver for more information.

### *SQLPrepare:*

This function prepares an IBM i Access ODBC SQL statement for execution.

In C, this statement is coded:

**Note:** SQL\_NTS indicates that the string is null-terminated.

```
SQLCHAR szSQLstr[ ] = "INSERT INTO NAMEID VALUES (?,?)";
```

```
rc = SQLPrepare(hstmt, szSQLstr, SQL_NTS);
```

### *SQLBindParameter:*

This function allows an IBM i Access ODBC 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:*

This function 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 and SQLPutData:*

These statements supply unbound, input parameter values when the IBM i Access ODBC SQL statement is executed.

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 and from an array of Byte data types and bind the array of Byte. This method is demonstrated below, in the Convert strings and arrays of byte topic.

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:

- These two statements operate together to supply unbound parameter values when the statement is executed.
- Each call to **SQLParamData** moves the internal pointer to the next parameter for **SQLPutData** to supply data. After the last parameter is filled, **SQLParamData** must be called again for the statement to be executed.
- 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.

'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

```

*Convert 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
```

*Performance architecture of the IBM i Access for Windows ODBC driver:*

For the IBM i Access ODBC driver, all of the internal data flows between the client and the server are chained together, and transmitted 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 configurable on the IBM i Access 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.

## ODBC API return codes:

Every IBM i Access 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.

## End ODBC functions:

The last procedure that must be completed before ending an IBM i Access 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);
```

## Implementation issues of ODBC APIs

Learn about implementations issues when using IBM i Access ODBC APIs.

Choose from the following topics for information regarding implementation of ODBC APIs.

**Note:** For a description and work-around for several problems that can occur when using the IBM i Access ODBC driver with Microsoft's ADO interface, search the Software Knowledge Base, using ADO Stored Procedure Calls with MSDASQL as a search string.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 578.

### Related reference

"Example: Run CL commands that use SQL stored procedures and ODBC" on page 565

Stored procedure support provides a means to run IBM i Control Language (CL) commands by using the SQL CALL statement.

### Related information

 Software Knowledge Base

For a description and work-around for several problems that can occur when using the IBM i Access ODBC driver support with Microsoft's ADO interface, search the Software Knowledge Base, using ADO Stored Procedure Calls with MSDASQL as a search string.

### ODBC 3.x API notes:

The following table lists IBM i Access ODBC 3.x APIs by their associated task and identifies considerations for each API.

### Notes:

- The IBM i Access 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 IBM i Access 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.
- For more details on how these APIs work, search for ODBC at the Microsoft Web site.

Type	API	Description	Other considerations
<b>Connecting to a data source</b>			
<b>Note:</b> For information on how the connection APIs prompt signon dialogs see "Signon dialog behavior" on page 514. Also see connection pooling for more information.			

Type	API	Description	Other considerations
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 <b>Connection options</b> dialog on the <b>General</b> tab of the DSN.
Char	SQLDriverConnect	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialogs for the user.	Uses all keywords. Only DSN is required. Other values are optional. Refer to "Connection string keywords" on page 497 for more information.
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 "Connection string keywords" on page 497 for more implementation issues.
<b>Get information regarding a driver or data source</b>			
Byte	SQLGetInfo	Returns information about a specific driver and data source.	<p>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:</p> <ul style="list-style-type: none"> <li>• SQL_CATALOG_NAME_SEPARATOR – By default a period is returned. If the connection string keyword NAM is set to 1, a comma is returned.</li> <li>• 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.</li> <li>• SQL_DATA_SOURCE_READ_ONLY – By default N is returned. If the connection string keyword CONNTYPE is set to 0 then Y is returned.</li> <li>• 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.</li> <li>• 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.</li> <li>• 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.</li> <li>• SQLDriverVer - Returns the version of the driver in the format of VV.RR.SSST, where, <ul style="list-style-type: none"> <li>– VV represents the version of the IBM i Access for Windows product.</li> <li>– RR is the release identifier of the IBM i Access for Windows product.</li> <li>– SSS is the number of the service pack that has been applied to the IBM i Access for Windows product, and</li> <li>– T is the version of the test fix that has been applied for an ODBC driver problem, otherwise, it is 0.</li> </ul> </li> </ul>

Type	API	Description	Other considerations
N/A	SQLGetTypeInfo	Returns information about supported data types.	<p>Different result data types can be seen when running to different IBM i versions. For example, the DECFLOAT data type is only in the result set when running to V6R1 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 for i database types" on page 515 for more information.</p>
<b>Set and retrieve driver attributes</b>			
<b>Note:</b> Refer to "Connection and statement attributes" on page 520 for details on driver-specific connection and statement attributes applicable to the following APIs.			
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. If working with LOB locator fields the driver restricts these values to 1 row at a time. LOB fields are handled as locators if the MAXFIELDLEN connection string value is less than the LOB field size.</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"> <li>• SQL_ATTR_ASYNC_ENABLE</li> <li>• SQL_ATTR_RETRIEVE_DATA</li> <li>• SQL_ATTR_SIMULATE_CURSOR</li> <li>• SQL_ATTR_USE_BOOKMARKS</li> <li>• SQL_ATTR_FETCH_BOOKMARK_PTR</li> <li>• SQL_ATTR_KEYSET_SIZE</li> </ul> <p>Setting SQL_ATTR_MAX_ROWS is supported, however, it only impacts performance for static cursors. The full result set is still built with other cursor types even if this option is set. Using the FETCH FIRST x ROWS ONLY clause in your SQL query may work better since it reduces the amount of work the server does. This API has been extended to also contain the cursor row count for the following two result set types:</p> <ul style="list-style-type: none"> <li>• stored procedure array result sets</li> <li>• static cursor result sets</li> </ul>

Type	API	Description	Other considerations
Byte	SQLGetStmtAttr	Returns the value of a statement option.	The following are not supported: <ul style="list-style-type: none"> <li>• SQL_ATTR_ASYNC_ENABLE</li> <li>• SQL_ATTR_RETRIEVE_DATA</li> <li>• SQL_ATTR_SIMULATE_CURSOR</li> <li>• SQL_ATTR_USE_BOOKMARKS</li> <li>• SQL_ATTR_FETCH_BOOKMARK_PTR</li> </ul>
<b>Set and retrieve descriptor fields</b>			
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.	Can not set descriptor fields for an IRD other than SQL_DESC_ARRAY_STATUS_PTR and SQL_DESC_ROWS_PROCESSED_PTR.  Does not support named parameters.
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.
<b>Prepare SQL requests</b>			
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. Set the UNICODESQL keyword equal 1 or equal 2, to enable the driver to send SQL statement text to the host in Unicode. Note that when sending Unicode SQL statements the driver generates 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>See SQL Statement Considerations for several SQL statements that are not recommended to be prepared and executed.</p> <p>For information on which escape sequences and scalar functions the driver supports see "SQLPrepare and SQLNativeSQL escape sequences and scalar functions" on page 523.</p>

Type	API	Description	Other considerations
Byte	SQLBindParameter	Assigns storage for a parameter in an SQL statement. See "Parameter markers" on page 473 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>On V6R1 and later hosts:</p> <ul style="list-style-type: none"> <li>SQL_DEFAULT_PARAM and SQL_UNASSIGNED are only valid in binding to an INSERT or UPDATE Statement. Use on other statements will return an error.</li> <li>If Strlen_or_IndPtr = SQL_DEFAULT_PARAM, the driver uses the column default value.</li> <li>If SQL_UNASSIGNED or Strlen_or_IndPtr = -7, and the statement is an UPDATE statement, the corresponding column value in the table being updated is unchanged. On an INSERT statement, the default value is used.</li> </ul> <p>On pre-V6R1 hosts:</p> <ul style="list-style-type: none"> <li>Default and unassigned parameters are not supported.</li> <li>If SQL_DEFAULT_PARAM or SQL_UNASSIGNED is specified for the Strlen_or_IndPtr parameter, the driver returns an error during the execution of the SQL statement.</li> <li>For binding statements using SQL_DEFAULT_PARAM or SQL_UNASSIGNED, the driver returns SQLSTATE error of 07S01.</li> </ul>
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, <b>myCursorName</b> becomes <b>MYCURSORNAME</b> while "myCursorName" is treated as <b>myCursorName</b>, with a length of 14 since the quotes are included in the length.</p> <p>The driver supports only these characters in cursor names: <b>"", a-z, A-Z, 0-9, or _</b>. 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 maximum cursor name is 128 characters, 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"> <li>Cursor name changes are not allowed during the DRDA connection.</li> <li>Cursor names will be changed by the driver and should be checked via SQLGetCursorName after the cursor is open. (after SQLExecute or SQLExecDirect).</li> </ul>
<b>Submit requests</b>			
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 497 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.	



Type	API	Description	Other considerations
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).	
<b>Retrieve results and related information</b>			
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 or an array result set.
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 cursors.	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 474 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.	<p>SQL_UPDATE, SQL_DELETE, SQL_ADD are unsupported options for Operations parameter.</p> <p>SQL_LOCK_EXCLUSIVE, SQL_LOCK_UNLOCK are unsupported options for the LockType parameter.</p>
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.	
<b>Get data source system table information</b>			
Char	SQLColumnPrivileges	Returns a list of columns and associated privileges for one or more tables.	
Char	SQLColumns	Returns a list of information on columns in one or more tables.	

Type	API	Description	Other considerations
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 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.	Beginning with V6R1, you can define a derived key index. When SQLStatistics is used to retrieve information about the index, the COLUMN_NAME result set column returns the expression that represents the derived key index.  If the WHERE clause was used when creating the index, the Where expression is returned in FILTER_CONDITION result set column.
Char	SQLTables	Returns a list of schemas, tables, or table types in the data source.	See "SQLTables Description" on page 529
Char	SQLTablePrivileges	Returns a list of tables and the privileges that are associated with each table.	
Char	SQLPrimaryKeys	Returns the list of column name or names that comprise the primary key for a table.	
<b>Clean up a statement</b>			
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. For more information, see "Handle long-running queries" on page 529.
N/A	SQLEndTran	Commits or rolls back a transaction.	For information regarding commitment control, see Commitment control considerations.
<b>Terminate a connection</b>			
N/A	SQLDisconnect	Closes the connection.	
N/A	SQLFreeHandle	Releases resources associated with handles.	

## Related reference

"ODBC API restrictions and unsupported functions" on page 514

The way in which some functions are implemented in the IBM i Access ODBC Driver does not meet the specifications in the Microsoft ODBC Software Development Kit Programmer's Reference.

## Related information

 Microsoft Web site

### SQL Statement Considerations:

Identify SQL statements to avoid when using ODBC with IBM i Access functions.

There are several SQL statements that are not recommended to be prepared and executed. Examples of these are:

- SET TRANSACTION

- SET SCHEMA
- SET PATH
- COMMIT
- ROLLBACK
- CONNECT TO
- DISCONNECT ALL

For these statements, you can accomplish the same behavior in other ways through ODBC. For example, if you turn off autocommit for the ODBC connection, you can use the `SQLEndTran` option instead of attempting to execute a `COMMIT` or `ROLLBACK` statement.

Note that the `SET SESSION AUTHORIZATION` SQL statement changes the user that is in control of that connection which leads to unpredictable behavior when used in combination with ODBC connection pooling. The recommended way to use the `SET SESSION AUTHORIZATION` statement, through ODBC, is to free all open statement handles except for the `SET SESSION AUTHORIZATION` on which it is to run. Once `SET SESSION AUTHORIZATION` is run, you should free the statement handle.

### Connection string keywords:

The IBM i Access support for the ODBC driver has many connection string keywords that are used to change the behavior of the ODBC connection.

These same keywords and their values are 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.

Choose from the following tables for more information on the connection string keywords that are recognized by the IBM i Access support for the ODBC driver.

#### *Connection string keywords - General properties:*

Use these IBM i Access ODBC driver connection string keywords to change General properties of the ODBC connection.

The following table lists connection string keywords for General properties that are recognized by the IBM i Access ODBC driver:

*Table 3. IBM i Access ODBC connection string keywords for General properties*

Keyword	Description	Choices	Default
DSN	Specifies the name of the ODBC data source that you want to use for the connection.	Data source (DSN) name	none

Table 3. IBM i Access ODBC connection string keywords for General properties (continued)

Keyword	Description	Choices	Default
DRIVER	Specifies the name of the ODBC driver that you want to use. <b>Note:</b> This should not be used if the DSN property has been specified.	"iSeries Access ODBC Driver"  Client Access ODBC Driver (32-bit) <b>Note:</b> For IBM i Access for Windows, two ODBC drivers are registered. Both the <b>Client Access ODBC Driver (32-bit)</b> and the <b>iSeries Access ODBC Driver</b> names are registered, however, both of these registered names point to the same ODBC driver. These two registered names do not indicate that two different ODBC drivers are installed. The older name of <b>Client Access ODBC Driver (32-bit)</b> is registered to support backward compatibility.	none
PWD or Password	Specifies the password for the IBM i user ID for the connection.	IBM i 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 System iNavigator default 4 = Use Kerberos principal	3
SSL	Specifies whether a Secure Sockets Layer (SSL) connection is used to communicate with the server.	0 = Encrypt only the password 1 = Encrypt all clients/server communication	0
SYSTEM	Specifies the IBM i system name to connect.	IBM i name. See IBM i name formats for ODBC Connection APIs.	none
UID or UserID	Specifies the user ID for the IBM i connection.	IBM i user ID	none

Connection string keywords - Server properties:

Use these IBM i Access ODBC driver connection string keywords to change Server properties of the ODBC connection.

The following table lists connection string keywords for Server properties that are recognized by the IBM i Access ODBC driver:

Table 4. IBM i Access ODBC connection string keywords for Server properties

Keyword	Description	Choices	Default
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

Table 4. IBM i Access ODBC connection string keywords for Server properties (continued)

Keyword	Description	Choices	Default
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
DATABASE	Specifies the IBM i relational database (RDB) name to connect.  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).	IBM i relational database name	empty-string
DBQ or DefaultLibraries	Specifies the IBM i 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.  <b>Note:</b> The first library listed in this property will also be the default schema (or library ), which is used to resolve unqualified names in SQL statements. To specify no default schema, a comma should be entered before any libraries.	IBM i libraries  75 entries are supported. Entries over 75 are ignored.	QGPL
MAXDECPREC or Maximum Decimal Precision	Specifies the maximum precision of decimal data that will be returned.	31 or 63	31
MAXDECSCALE or Maximum Decimal Scale	Specifies the maximum scale used in arithmetic calculations involving decimal data. This value must be less than the value of MAXDECPREC.	0 – 63	31
MINDIVSCALE or Minimum Divide Scale	Specifies the minimum scale used in arithmetic calculations involving decimal data.	0 – 9	0
NAM or Naming	Specifies the naming convention used when referring to tables.	0 = "sql" (as in <i>schema.table</i> )  1 = "system" (as in <i>schema/table</i> )	0

Connection string keywords - Data types:

Use these IBM i Access ODBC driver connection string keywords to change the Data types properties of the ODBC connection.

The following table lists connection string keywords for the Data types properties that are recognized by the IBM i Access ODBC driver:

*Table 5. IBM i Access ODBC connection string keywords for the Data types properties*

<b>Keyword</b>	<b>Description</b>	<b>Choices</b>	<b>Default</b>
DFT or DateFormat	Specifies the date format used in date literals within SQL statements.	0 = yy/dd (*JUL) 1 = mm/dd/yy (*MDY) 2 = dd/mm/yy (*DMY) 3 = yy/mm/dd (*YMD) 4 = mm/dd/yyyy (*USA) 5 = yyyy-mm-dd (*ISO) 6 = dd.mm.yyyy (*EUR) 7 = yyyy-mm-dd (*JIS)	5
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).	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.	0 = "." (period) 1 = "," (comma)	0
DECFLOATERROROPTION	Specifies whether a warning or data mapping error is reported when encountering an error with the decimal floating point data type. If not provided, the server attribute value is unchanged.	0 = Report decimal floating point error as a data mapping error 1 = Report decimal floating point error as a warning	0

Table 5. IBM i Access ODBC connection string keywords for the Data types properties (continued)

Keyword	Description	Choices	Default
DECFLOATROUNDMODE	Specifies the rounding mode, when rounding is allowed for a result.	<p>0 = ROUND_HALF_EVEN - round to nearest digit. If equidistant, round to the nearest even digit. This is the default rounding mode.</p> <p>1 = ROUND_HALF_UP - round to nearest digit. If equidistant, round up.</p> <p>2 = ROUND_DOWN - round to nearest lower digit. This is the same as truncation.</p> <p>3 = ROUND_CEILING - round towards +infinity.</p> <p>4 = ROUND_FLOOR - round towards -infinity.</p> <p>5 = ROUND_HALF_DOWN - round to nearest digit. If equidistant, round down.</p> <p>6 = ROUND_UP - round to nearest higher digit.</p>	0
MAPDECIMAL FLOATDESCRIBE	Specify the format for the results of a DECFLOAT operation. <b>Note:</b> Merge the three MAPDECIMAL and FLOATDESCRIBE strings into a single string before using.	<p>1 = SQL_VARCHAR</p> <p>3 = SQL_DOUBLE</p>	1
TFT or TimeFormat	Specifies the time format used in time literals within SQL statements.	<p>0 = hh:mm:ss (*HMS)</p> <p>1 = hh:mm AM/PM (*USA)</p> <p>2 = hh.mm.ss (*ISO)</p> <p>3 = hh.mm.ss (*EUR)</p> <p>4 = hh:mm:ss (*JIS)</p>	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".	<p>0 = ":" (colon)</p> <p>1 = "." (period)</p> <p>2 = "," (comma)</p> <p>3 = " " (blank)</p>	0
XMLSTRIPWS or CurrentImplicitXmlParseOption	Specifies the XMLPARSE option to use for the connection. This attribute indicates how whitespace in serialized XML data should be handled by DB2 when the data is implicitly parsed without validation.	<p>0 = STRIP WHITESPACE</p> <p>1 = PRESERVE WHITESPACE</p>	0

Table 5. IBM i Access ODBC connection string keywords for the Data types properties (continued)

Keyword	Description	Choices	Default
XMLDECLARATION or XMLDeclaration	Specifies the XML Declaration to return with XML columns that are returned in result sets.	<p>0 = No declarations or byte order marks (BOMs) are added to the output buffer.</p> <p>1 = A byte order mark (BOM) in the appropriate endianness is prepended to the output buffer if the target encoding is UTF-16.</p> <p>2 = A minimal XML declaration is generated, containing only the XML version.</p> <p>4 = An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this attribute.</p> <p>7 = An encoding attribute that indicates that a BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.</p>	7

Connection string keywords - Package properties:

Use these IBM i Access ODBC driver connection string keywords to change Package properties of the ODBC connection.

The following table lists connection string keywords for Package properties that are recognized by the IBM i Access ODBC driver:

Table 6. IBM i Access ODBC connection string keywords for the Package properties

Keyword	Description	Choices	Default
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



Table 6. IBM i Access ODBC connection string keywords for the Package properties (continued)

Keyword	Description	Choices	Default
PKG or DefaultPackage	<p>Specifies how the extended dynamic (package) support will behave. The string for this property must be in the following format: <b>A/DEFAULT(IBM),x,0,y,z,0</b></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"> <li>• x = Specifies whether or not to add statements to an existing SQL package.</li> <li>• 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.</li> <li>• 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.</li> </ul> <p><b>Note:</b> This property has no effect unless the XDYNAMIC property is set to 1.</p>	<p><b>A/DEFAULT(IBM),x,0,y,z,0</b></p> <p>Values for x option:</p> <ul style="list-style-type: none"> <li>• 1 = Use (Use the package, but do not put any more SQL statements into the package)</li> <li>• 2 = Use/Add (Use the package and add new SQL statements into the package)</li> </ul> <p>Values for y option:</p> <ul style="list-style-type: none"> <li>• 0 = Return an error (SQL_ERROR) to the application</li> <li>• 1 = Return a warning (SQL_SUCCESS_WITH_INFO) to the application</li> <li>• 2 = Return success (SQL_SUCCESS) to the application</li> </ul> <p>Values for z option:</p> <ul style="list-style-type: none"> <li>• 0 = Do not cache SQL package locally</li> <li>• 1 = Cache SQL package locally</li> </ul>	default
XDYNAMIC or ExtendedDynamic	<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><b>Note:</b> For more information see “Use Extended Dynamic SQL” on page 533.</p>	<p>0 = Disable extended dynamic support</p> <p>1 = Enable extended dynamic support</p>	1

**Note:** **A/DEFAULT(IBM),x,0,y,z,0** is the default value for PKG or DefaultPackage.

*Connection string keywords - Performance properties:*

Use these IBM i Access ODBC driver connection string keywords to change Performance properties of the ODBC connection.

The following table lists connection string keywords for Performance properties that are recognized by the IBM i Access ODBC driver:

Table 7. IBM i Access ODBC connection string keywords for Performance properties

Keyword	Description	Choices	Default
BLOCKFETCH	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. <b>Note:</b> For more information on setting this option see the Fine-tuning record blocking topic.	0 = Use ODBC settings for blocking  1 = Use blocking with a fetch of 1 row	1
BLOCKSIZE or BlockSizeKB	Specifies the block size (in kilobytes) that is retrieved on FETCH requests and then cached 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.	1 – 8192	256
COMPRESSION or AllowDataCompression	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.	0 = Disable compression  1 = Enable compression	1
CONCURRENCY	Specifies whether to override the ODBC concurrency setting by opening all cursors as updateable. <b>Note:</b> In the following two cases, setting this option has no effect:  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.	0 = Use ODBC concurrency settings  1 = Open all cursors as updateable	0
CURSORENSITIVITY	Specifies the cursor sensitivity to use when opening cursors. This option applies to all forward-only and dynamic cursors that are opened on the same connection. Static cursors are always insensitive.	0 - Unspecified/Asensitive  1 = Insensitive  2 = Sensitive	

Table 7. IBM i Access ODBC connection string keywords for Performance properties (continued)

Keyword	Description	Choices	Default
EXTCOLINFO or ExtendedColInfo	<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"> <li>• SQL_DESC_AUTO_UNIQUE_VALUE</li> <li>• SQL_DESC_BASE_COLUMN_NAME</li> <li>• SQL_DESC_BASE_TABLE_NAME and SQL_DESC_TABLE_NAME</li> <li>• SQL_DESC_LABEL</li> <li>• SQL_DESC_SCHEMA_NAME</li> <li>• SQL_DESC_SEARCHABLE</li> <li>• SQL_DESC_UNNAMED</li> <li>• SQL_DESC_UPDATABLE</li> </ul> <p><b>Note:</b> 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 for i 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. <b>Note:</b> This option can cause problems due to the cursors still holding locks on the result set rows after the close request.</p>	<p>0 = Close all cursors immediately</p> <p>1 = Delay closing of cursors until the next request</p>	0

Table 7. IBM i Access ODBC connection string keywords for Performance properties (continued)

Keyword	Description	Choices	Default
MAXFIELDLEN or MaxFieldLength	<p>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.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>Setting this property to 0 forces the driver to always retrieve the LOB values with additional communication flows.</li> <li>Setting this property larger than 15360 KB has no effect. Anything larger than 15360 KB is retrieved in pieces from the server. Retrieving the data in pieces reduces the amount of memory needed, at any given time, on the client.</li> </ul>	0 — 2097152	32
PREFETCH	Specifies whether to prefetch data upon executing a SELECT statement. This increases performance when accessing the initial rows in the ResultSet.	0 = Do not prefetch data 1 = Prefetch data	1
QRYSTGLMT	Specifies storage limit for a query. If the estimated storage usage exceeds the specified storage limit in the parameter, the query is not executed.	*NOMAX = No Query Limit 0 - 2147352578	*NOMAX
QUERYOPTIMIZEGOAL	Specifies the optimization goal for queries. This parameter corresponds to the QAQQINI option called OPTIMIZATION_GOAL. For more information, refer to the QAQQINI option in the DB2 for i SQL Reference.	0 = Use the goal of *ALLIO if extended dynamic support is enabled, otherwise use the *FIRSTIO goal. 1 = *FIRSTIO - Return the first block of data as fast as possible. 2 = *ALLIO - Optimize as if the complete result set will be read by the application.	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

## Related reference

“Fine-tune record-blocking” on page 533

**Record-blocking** is a technique that significantly reduces the number of network flows and therefore improves performance when using IBM i Access ODBC driver.

### Connection string keywords - Language properties:

Use these IBM i Access ODBC driver connection string keywords to change the Language properties of the ODBC connection.

The following table lists connection string keywords for Languages that are recognized by the IBM i Access ODBC driver:

Table 8. IBM i Access ODBC connection string keywords for the Language properties

Keyword	Description	Choices	Default
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, SQI, SRB, SRL, SVE, THA, TRK, UKR, URD, VIE	ENU
SORTTABLE	Specifies the library and file name of a sort sequence table stored on the system. 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 or 1 = Sort based on hexadecimal values 2 = Sort based on the language set in LANGUAGEID property 3 = Sort based on the sort sequence table set in the SORTTABLE property	0
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

### Connection string keywords - Catalog properties:

Use these IBM i Access ODBC driver connection string keywords to change Catalog properties of the ODBC connection.

The following table lists connection string keywords for Catalog properties that are recognized by the IBM i Access ODBC driver:

Table 9. IBM i Access ODBC connection string keywords for the Catalog properties

Keyword	Description	Choices	Default
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.	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 = i5/OS 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

**Note:** "A/DEFAULT(IBM),x,0,y,z,0" is the default value for PKG or DefaultPackage.

*Connection string keywords - Conversion properties:*

Use these IBM i Access ODBC driver connection string keywords to change Conversion properties of the ODBC connection.

The following table lists connection string keywords for Conversion properties that are recognized by the IBM i Access ODBC driver:

Table 10. IBM i Access ODBC connection string keywords for Conversion properties

Keyword	Description	Choices	Default
ALLOWUNSCCHAR or AllowUnsupportedChar	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 10. IBM i Access ODBC connection string keywords for Conversion properties (continued)

Keyword	Description	Choices	Default
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. This property affects two different behaviors: 1. Whether the length of a graphic field is reported as a character count or as a byte count by the SQLDescribeCol API. 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
HEXPARSEROPT or Hex Parser Option	Specifies how SQL hexadecimal constants will be interpreted in SQL statements.	0 = Treat hexadecimal constants as character data  1 = Treat hexadecimal constants as binary data	0
TRANSLATE or ForceTranslation	Specifies whether or not to convert binary data (CCSID 65535) to text. When this property is set to 1, binary fields are treated as character fields.	0 = Do not convert binary data to text  1 = Convert binary data to text	0
UNICODESQL	Specifies whether or not to send Unicode SQL statements to the server.	0 = Send EBCDIC SQL statements to the server  1 = Send UCS-2 Unicode SQL statements to the server in UCS-2	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

*Connection string keywords - Diagnostic properties:*

Use these IBM i Access ODBC driver connection string keywords to change Diagnostic properties of the ODBC connection.

The following table lists connection string keywords for Diagnostic properties that are recognized by the IBM i Access ODBC driver:

Table 11. IBM i Access ODBC connection string keywords for Diagnostic properties

Keyword	Description	Choices	Default
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 QRYOPTLIB 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 for i SQL diagnostic options to be set. Use only as directed by your technical support provider.	DB2 for i 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  2 = Enable Database Monitor  4 = Enable the Start Debug (STRDBG) command  8 = Print job log at disconnect  16 = Enable job trace  32 = Enable database host server trace	0

Connection string keywords - other properties:

Use these IBM i Access ODBC driver connection string keywords to change other properties of the ODBC connection.

The following table lists other connection string keywords that are recognized by the IBM i Access ODBC driver:

Table 12. IBM i Access ODBC connection string keywords for other properties

Keyword	Description	Choices	Default
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



Table 12. IBM i Access ODBC connection string keywords for other properties (continued)

Keyword	Description	Choices	Default
CONCURRENT ACCESS RESOLUTION	Contains the preference for concurrent access resolution. This property identifies how a row lock conflict should be handled when it is encountered. This property only applies to read-only queries with isolation level CS. <b>Note:</b> Merge the three CONCURRENT, ACCESS, and RESOLUTION strings into a single string before using.	0 = Use Server Setting 1 = Use Currently Committed Rows 2 = Wait for Outcome 3 = Skip Locks	0
DB2SQLSTATES	Specifies whether or not to return ODBC-defined SQL States or DB2 SQL States. Refer to the DB2 for i 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
DATETIMETOCHAR or ConvertDateTimeToChar	Specifies one or more options on how date, time, and timestamp data types are reported to an application. To specify multiple options add together the values for the options. This option supports cases in which date values such as 24:00:00 are used.	To determine the value for this keyword, add the values below that are associated with each option that you want.  0 = Continue to map the DATE, TIME, and TIMESTAMP data types as SQL_TYPE_DATE, SQL_TYPE_TIME, and SQL_TYPE_TIMESTAMP  1 = Return DATE data type as SQL_CHAR  2 = Return TIME data type as SQL_CHAR  4 = Return TIMESTAMP data type as SQL_CHAR	0
DBCSNoTruncError	Specifies whether or not to report a DBCS string conversion overflow error as an ODBC truncation error.	0 = Report DBCS string conversion overflow error as ODBC truncation error  1 = Ignore truncation error	0

Table 12. IBM i Access ODBC connection string keywords for other properties (continued)

Keyword	Description	Choices	Default
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 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> <p>128 = Subtract one from the SQLBindParameter sourcelength if the last character in the buffer is a null-terminator character.</p> <p>256 = Ignore data decimal errors</p> <p>512 = Ignore cast warnings (SQL0402) for scrollable cursors</p> <p>1024 = Disable variable length compression</p> <p>2048 = Return no support for SQL_CVT_DATE when calling the SQLGetInfo's SQL_CONVERT_TIMESTAMP option.</p> <p>32768 = If the result of a query results in a column being divided by 0, return a NULL value instead of an error.</p>	0

Table 12. IBM i Access ODBC connection string keywords for other properties (continued)

Keyword	Description	Choices	Default
TRUEAUTOCOMMIT	Specifies how to handle autocommit support. In past ODBC drivers, turning autocommit on resulted in the server running under the *NONE isolation level. Now autocommit can run under any isolation level. Applications that require strict conformance to the SQL specification should use setting 1. Note that this setting requires that all files be journaled. Setting 0 offers better performance for most applications. See the SQL reference for further information on Transaction Isolation levels.	0 = run autocommit under the *NONE isolation level  1 = run autocommit under the isolation level that is set for the connection. The connection's isolation level is set using the SQLSetConnectAttr API and the SQL_ATTR_TXN_ISOLATION option.	0
NEWPWD	Specifies a new password used to override the current user's IBM i password. This option is only honored if set by an application. When using this option, the UID and PWD keywords should also be specified.	New password to use	none
XALCS or XALooselyCoupledSupport	Specifies whether locks are shared between loosely coupled distributed transaction branches.	0 = Locks are not shared  1 = Locks are shared	1 (for Windows)  0 (for Linux)
XALOCKTIMEOUT	Specifies the maximum amount of time (in seconds) that a distributed transaction waits on a lock request before timing out.	0 = Use the default system setting  0 – 999999999 = the number of seconds to wait	0
XATXNTIMEOUT or XATransactionTimeout	Specifies the amount of time (in seconds) that a distributed transaction waits before timing out.	0 = Wait indefinitely for the transaction to finish  0 – 999999999 = the number of seconds to wait	0

### Version and release changes in the ODBC driver behavior:

This topic describes new features supported by different versions of the ODBC driver and corresponding IBM i release.

#### The following list describes some of the important changes for 7.1:

New support include the following when using the ODBC driver to access 7.1 IBM i data:

- XML data type
- 128-byte schema names
- Multiple-row UPDATE, DELETE, and MERGE statements
- Concurrent Access Resolution support

**The following list describes some of the important changes for V6R1:**

New support include the following when using the ODBC driver to access V6R1 System i data:

- SQL query storage limits
- ODBC application and QZDASOINIT system jobs association
- 128-byte cursor names
- Decimal Floating Point (DECFLOAT) data type
- Additional stored procedure date and time formats

**The following list describes some of the important changes for V5R4:**

There are several new features available when using the ODBC driver to access V5R4 System i data. These features include the following.

- Support for 128-byte column names
- Support for longer SQL statements (commands can be up to 2,097,152 bytes or 1,048,576 characters long)
- Support for passing an IBM Enterprise Workload Manager™ (eWLM) correlator to the i5/OS host
- Improved support for table and column names that are not all uppercase
- Enhanced distributed transaction support for loosely coupled transactions
- A Linux 64-bit ODBC driver

**ODBC API restrictions and unsupported functions:**

The way in which some functions are implemented in the IBM i Access 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 API notes” on page 490 for a list of individual APIs and their associated considerations.

*Table 13. 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.

**Related reference**

“ODBC 3.x API notes” on page 490

The following table lists IBM i Access ODBC 3.x APIs by their associated task and identifies considerations for each API.

**Signon dialog behavior:**

You have control over your IBM i Access for Windows signon dialog, userid, and password prompting.

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 **Connection Options** 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 the System iNavigator 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.

### ActiveX Data Objects (ADO) prompting

When coding your ODBC application using ActiveX Data Objects (ADO) the default behavior for prompting is **adPromptNever**. To prompt differently, set the Prompt property on the Connection object prior to calling the Connection's Open method. For example, the following ADO code would result in prompting only as needed. Adding the SIGNON, UID, or PWD keywords allow you to have more control over the amount of prompting.

```
Dim conn As New ADODB.Connection
conn.Properties("Prompt") = adPromptComplete
conn.Open "Provider = MSDASQL;DSN=myODBCDSN;
```

### ODBC data types and how they correspond to DB2 for i database types:

The IBM i Access support for the ODBC driver maps data types between ODBC types and DB2 for i data types.

The following table shows the mappings of the supported data types. Choose the related link below, to the DB2 for i database types, for more information on data types.

Table 14. Data Type Mapping for DB2 for i database types

3.x ODBC Data Type	DB2 for i Database Type
SQL_BIGINT	BIGINT
SQL_BINARY	BINARY or XML 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
SQL_VARBINARY	VARBINARY VARCHAR FOR BIT DATA LONG VARCHAR FOR BIT DATA ROWID
SQL_VARCHAR	VARCHAR VARGRAPHIC LONG VARCHAR LONG VARGRAPHIC DATALINK DECFLOAT
SQL_WCHAR	GRAPHIC CCSID 1200 GRAPHIC CCSID 13488
SQL_WLONGVARCHAR	DBCLOB CCSID 1200 DBCLOB CCSID 13488
SQL_WVARCHAR	VARGRAPHIC CCSID 1200 VARGRAPHIC CCSID 13488 LONGVARGRAPHIC CCSID 1200 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.
- Unicode data which are the data types with a CCSID of 1200 (UTF-16), CCSID of 1208 (UTF-8), or 13488 (UCS-2), report to ODBC 2.x applications as `SQL_CHAR`, `SQL_VARCHAR`, and `SQL_LONGVARCHAR` instead of `SQL_WCHAR`, `SQL_WVARCHAR`, and `SQL_WLONGVARCHAR`.
- LOBs (`BLOB`, `CLOB`, and `DBCLOB`) up to 2 GB in size are supported. For more information on LOBs and datalinks choose the related link below, to the Large Objects (LOBs) considerations topic collection.
- Note that to retrieve decimal fields with large precision successfully you must bind the column as `SQL_C_CHAR`. The structure that stores `SQL_C_NUMERIC` data can hold up to 38 digits.

#### Related reference

"Large objects (LOBs) considerations" on page 519

Use LOBs with IBM i Access ODBC to store and access large text documents.

#### Related information

DB2 for i database types

#### Working with the XML data type:

- | These conventions can help you handle various aspects of using the XML data type in DB2 for iODBC
- | functions.

#### XML data handling in ODBC applications

DB2 for i ODBC applications can retrieve and store XML data using the `SQL_XML` data type. This data type corresponds to the native XML data type of the DB2 for i database, which is used to define columns that store well-formed XML documents. The `SQL_XML` type can be bound to the following C types: `SQL_C_BINARY`, `SQL_VARBINARY`, `SQL_C_CHAR`, `SQL_VARCHAR`, `SQL_C_WCHAR`, and `SQL_WVARCHAR`. Using binary types, however, instead of character types, is recommended to avoid possible data loss or corruption resulting from CCSID conversion when character types are used. To store XML data in an XML column, bind a binary (`SQL_C_BINARY` or `SQL_VARBINARY`) or character (`SQL_C_CHAR`, `SQL_VARCHAR`, `SQL_C_WCHAR`, or `SQL_WVARCHAR`) buffer that contains the XML value to the `SQL_XML` SQL type and execute the `INSERT` or `UPDATE` SQL statements. To retrieve XML data from the database, bind the result set to a binary (`SQL_C_BINARY` or `SQL_VARBINARY`) or character (`SQL_C_CHAR`, `SQL_VARCHAR`, `SQL_C_WCHAR`, or `SQL_WVARCHAR`) type. Use character types with caution because of encoding issues. When an XML value is retrieved into an application data buffer, the DB2 for i server performs an implicit serialization on the XML value to convert it from its internal form to the serialized string form. For character typed buffers, the XML value is implicitly serialized to the application CCSID associated with the character type. By default, an XML declaration is included in the output serialized string. This default behavior can be changed by setting the `SQL_ATTR_XML_DECLARATION` connection attribute.

#### XML column inserts and updates in ODBC applications

When you update or insert data into XML columns of a table, the input data must be in the serialized string format. For XML data, when you use `SQLBindParameter()` to bind parameter markers to input data buffers, you can specify the data type of the input data buffer as `SQL_C_BINARY`, `SQL_VARBINARY`, `SQL_C_CHAR`, `SQL_VARCHAR`, `SQL_C_WCHAR`, or `SQL_WVARCHAR`. When you bind a data buffer

that contains XML data as `SQL_C_BINARY` or `SQL_VARBINARY`, DB2 for i ODBC processes the XML data as internally encoded data. This method is preferred because it avoids the added processing and potential data loss of character conversion when character types are used. When you bind a data buffer that contains XML data as `SQL_C_CHAR`, `SQL_VARCHAR`, `SQL_C_WCHAR`, or `SQL_WVARCHAR`, DB2 for i ODBC processes the XML data as externally encoded data.

DB2 for i ODBC determines the encoding of the data as follows:

- If the C type is `SQL_C_WCHAR` or `SQL_WVARCHAR`, ODBC assumes that the data is encoded as UCS-2.
- If the C type is `SQL_C_CHAR` or `SQL_VARCHAR`, ODBC assumes that the data is encoded in the job CCSID.

The following example shows how to update XML data in an XML column using the recommended `SQL_C_BINARY` type.

```
char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);
SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                 xmlBuffer, 10240, &length); SQLExecute (hStmt);
```

### XML data retrieval in ODBC applications

When you select data from XML columns in a table, the output data is in the serialized string format. For XML data, when you use `SQLBindCol()` API to bind columns in a query result set to application variables, you can specify the data type of the application variables as `SQL_C_BINARY`, `SQL_VARBINARY`, `SQL_C_CHAR`, `SQL_VARCHAR`, `SQL_C_WCHAR`, or `SQL_WVARCHAR`. When retrieving a result set from an XML column, it is recommended that you bind your application variable to the `SQL_C_BINARY` or `SQL_VARBINARY` type. Binding to character types can result in possible data loss resulting from CCSID conversion. Data loss can occur when characters in the source code page cannot be represented in the target code page. Binding your variable to the binary types avoids these issues. XML data is returned to the application as internally encoded data.

ODBC determines the encoding of the data as follows:

- If the C type is `SQL_C_BINARY` or `SQL_VARBINARY`, DB2 for i ODBC returns the data in the encoding of the column.
- If the C type is `SQL_C_CHAR` or `SQL_VARCHAR`, DB2 for i ODBC returns the data in job CCSID.
- If the C type is `SQL_C_WCHAR` or `SQL_WVARCHAR`, DB2 for i ODBC returns the data in the UCS-2 encoding scheme.

The database server performs an implicit serialization of the data before returning it to the application. You can explicitly serialize the XML data to a specific data type by calling the `XMLSERIALIZE` function. Implicit serialization is recommended, however, because explicitly serializing to character types with `XMLSERIALIZE` can introduce encoding issues.

The following example shows how to retrieve XML data from an XML column into a binary application variable.

```
char xmlBuffer[10240];
// xmlBuffer is used to hold the retrieved XML document
integer length;
```



```
// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

length = sizeof (xmlBuffer);
SQLExecute (hStmt, "SELECT deptdoc FROM dept WHERE id='001'", SQL_NTS);
SQLBindCol (hStmt, 1, SQL_C_BINARY, xmlBuffer, &length, NULL);
SQLFetch (hStmt);
SQLCloseCursor (hStmt);
// xmlBuffer now contains a valid XML document encoded in UTF-8
```

### Large objects (LOBs) considerations:

Use LOBs with IBM i Access ODBC to store and access large text documents.

### Large objects (LOBs):

Large object (LOB) data types allow applications to store large data objects as strings. The ODBC driver can access LOBs that are up to 2 GB in size.

When uploading large LOB data fields to the server, it is recommended that you use the SQLParamData and SQLPutData APIs. The SQLPutData API sends the LOB data to the server as it is received and reduces the amount of memory needed on the client.

### 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 the Example: Use the BLOB data type topic below.

### For more information on LOBs:

See the Using large objects topic under the **Using the Object-Relational Capabilities** heading in the SQL Programming Concepts topic in the IBM i Information Center.

### 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 the Using DataLinks topic under the **Processing special data types** heading in the SQL Programming Concepts topic in the i5/OS Information Center.

### Related reference

“ODBC data types and how they correspond to DB2 for i database types” on page 515

The IBM i Access support for the ODBC driver maps data types between ODBC types and DB2 for i data types.

### Related information

SQL Programming Concepts

*Example: Use the BLOB data type:*

This is an example of using DB2 for IBM i BLOB data type with ODBC.

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 character column and a BLOB column
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', ...
    sprintf(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"

```

### Connection and statement attributes:

The IBM i Access ODBC specification defines several connection and statement attributes.

This ODBC specification is extended with several IBM i Access for Windows customized attributes, that are described in the following 2 tables.

Table 15. Customized connection attributes

Attribute	Get/Set	Description
1204	both	An unsigned value that controls the cursor commit behavior and cursor rollback behavior. Possible values: <ul style="list-style-type: none"> <li>• 0 - SQL_CB_DELETE is returned for SQLGetInfo's SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR options.</li> <li>• 1 - (default) SQL_CB_PRESERVE is returned for SQLGetInfo's SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR options.</li> </ul>
1281	both	Specifies the Client User ID string that is sent to the host database. This attribute is set after connected to a database. The maximum length is 255 characters. As an alternative, the CLIENTUSERID connection string keyword is used.
1282	both	Specifies the Work Station Name string that is sent to the host database. The maximum length is 255 characters. The attribute is set after connected to a database. As an alternative, the CLIENTWRKSTNNAME connection string keyword is used.
1283	both	Specifies the Application Name string using the ODBC driver. The maximum length for this attribute is 255 characters. This attribute is set after connected to a database. As an alternative, the CLIENTAPPLNAME connection string keyword is used.
1284	both	Specifies the Accounting ID string that is sent to the host database. The maximum length is 255 characters. This attribute is set after connected to the host database. As an alternative, the CLIENTACCTSTR connection string keyword is used.
2100	both	Can be used as an alternative to using the DFTPKGLIB 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.
2101	both	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: <ul style="list-style-type: none"> <li>• 0 - (default) A cell in a result set that contains a value calculated by dividing by zero will be returned as an error.</li> <li>• 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.</li> </ul>
2106	both	An alternative to using the COMPRESSION connection string keyword. This is an unsigned integer value. Possible values: <ul style="list-style-type: none"> <li>• 0 = (default) compression off</li> <li>• 1 = compression on</li> </ul>
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: <ul style="list-style-type: none"> <li>• 0 - (default) - don't trim CHAR fields</li> <li>• 1 - trim CHAR fields</li> </ul>
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: <ul style="list-style-type: none"> <li>• 10 character job name,</li> <li>• 10 character user,</li> <li>• 6 character job</li> </ul>

Table 15. Customized connection attributes (continued)

Attribute	Get/Set	Description
2116	set	A pointer to a buffer containing the IBM Enterprise Workload Manager (eWLM) correlator. Specifying this attribute allows you to tie your application with the eWLM support (Enterprise Workload Manager).
2117	both	Can be used as an alternative to the CONCURRENTACCESSRESOLUTION connection string keyword. This is an unsigned integer value which controls how conflicting row locks encountered in the transaction should be handled. Note: This property only applies to read-only queries with isolation level CS. <ul style="list-style-type: none"> <li>• 0 - (default) Use server setting</li> <li>• 1 - Use Currently Committed Rows</li> <li>• 2 - Wait for Outcome</li> <li>• 3 - Skip Locks</li> </ul>
2140	both	An unsigned integer value that specifies the amount of time (in seconds) that a distributed transaction waits before timing out. A value of 0 indicates to wait indefinitely for the transaction to finish. Setting this overrides any value that was set for the XATIMEOUT connection string keyword. The default value for this attribute is 0.
2141	both	An unsigned integer value that specifies the maximum amount of time (in seconds) that a distributed transaction waits on a lock request before timing out. A value of 0 indicates to use the default system settings. Setting this overrides any value that was set for the XALOCKTIMEOUT connection string keyword. The default value for this attribute is 0.
2142	both	An integer value that specifies the RMID to use for XA transaction work. This can be set at anytime. The RMID that is set must be unique for the process. If this value is set to 0 it indicates that any current XA transaction work has been completed for this connection. The default value for this attribute is 0.
2143	get	A character string that identifies the IBM i Access driver to call into for XA calls. This string is only valid if the 2142 connection attribute has been set. This string is set after the connection is established. The default value for this attribute is an empty-string.
2145	both	Can be used as an alternative to the XMLDECLARATION connection string keyword. This is an unsigned integer value that is used to indicate what type of XML declaration should be included with XML columns returned in result sets. Here are the values and the meanings: <ul style="list-style-type: none"> <li>• 0 - No declarations or byte order marks (BOMs) are added to the output buffer.</li> <li>• 1 - A byte order mark (BOM) in the appropriate endianness is prepended to the output buffer if the target encoding is UTF-16.</li> <li>• 2 - A minimal XML declaration is generated, containing only the XML version.</li> <li>• 4 - An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this attribute.</li> <li>• 7 - An encoding attribute that indicates that a BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.</li> </ul>
2146	both	Can be used as an alternative to the XMLSTRIPWS connection string keyword. This is an unsigned integer value that indicates how whitespace in serialized XML data should be handled by DB2 when the data is implicitly parsed without validation. indicates whether whitespace should be included or stripped from with the following possible values: <ul style="list-style-type: none"> <li>• 0 - STRIP WHITESPACE</li> <li>• 1 - PRESERVE WHITESPACE</li> </ul>

Table 15. Customized connection attributes (continued)

Attribute	Get/Set	Description
2511	both	Specifies the Program ID string that is sent to the host database. The maximum size is 255 characters. The attribute is set after connected to a database. As an alternative, the CLIENTPROGRAMID connection string keyword is used.

Table 16. Customized 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 are: <ul style="list-style-type: none"> <li>• 0 = compression off,</li> <li>• 1 = compression on</li> </ul>
2114	get	Returns an unsigned integer value that indicates the offset into a SQL statement at which a SQL syntax error has occurred. This will be set when SQLExecute or SQLExecDirect returns an SQL_ERROR return code.

### Connection pooling:

Connection pooling is supported on an IBM i Access ODBC connections.

Connection pooling refers to the behavior where IBM i Access ODBC connections are left open after the application has requested to disconnect them. Connections that are in the pool can be reused by the same application avoiding the time consuming operation of creating a brand new connection.

To get an application to use connection pooling support with the IBM i Access ODBC driver two basic steps must be taken:

1. Connection pooling support has to be enabled for the driver. To enable this support, open ODBC Administrator, click on the Connection Pooling tab, double-click on Client Access ODBC Driver (32-bit), and switch the checkbox to indicate to pool connections to this driver. There is also a spot on the window to fill in the amount of time that unused connections remain in the pool.
2. Connection pooling support has to be enabled by the application. An application does this by setting the SQL\_ATTR\_CONNECTION\_POOLING environment attribute as part of the connection process.

For more details on connection pooling support, search for ODBC at the Microsoft Web site.

### Related information

 Microsoft Web site

### SQLPrepare and SQLNativeSQL escape sequences and scalar functions:

The IBM i Access ODBC support includes escape sequences and scalar functions.

ODBC has escape sequences and scalar functions that are 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 IBM i Access for Windows ODBC driver.

### Escape sequences:

- d
- t
- ts
- escape
- oj
- call
- ?=call – This escape sequence should be used when trying to take advantage of the DB2 for IBM i support for return values from a procedure. The parameter marker will need to be bound as an output parameter using the SQLBindParameter API. Note, at this time procedures can only return values of type integer.

### Distributed transaction support:

Distributed transactions allow an IBM i Access ODBC application to coordinate units of work across multiple databases.

There are two different interfaces into the ODBC driver that allow one to complete a distributed transaction. The two interfaces are MTS (Microsoft Transaction Server) and XA API support. Both of these interfaces are affected by the setting of the XALOCKTIMEOUT and XATXNTIMEOUT connection string settings.

#### MTS

For more information on MTS refer to Using Distributed transaction support.

#### XA API support

Refer to the 2140, 2141, 2142, and 2143 connection attributes on the **Connection and statement attributes** page for a description of some of the relevant options for getting the XA support to work. Note, that the 2141 and 2142 connection attributes do the same thing as the XALOCKTIMEOUT and XATXNTIMEOUT connection string settings.

#### Note:

- xa\_open is only called by the application for recovery purposes. When connecting through the ODBC API SQLConnect or SQLDriverConnect the xa\_open is done automatically if the RMID was set via the 2142 connection attribute.
- The connection attribute SQL\_ATTR\_AUTOCOMMIT must be set as SQL\_AUTOCOMMIT\_ON.
- If an application wishes to start an XA transaction and then do some non-XA transaction work, one must set the RMID to 0 to indicate to the driver that the XA work is completed.
- To do XA recovery an application calls xa\_open with a string of:  
*SYSTEM=mySystem;UID=myUserID;PWD="myPassword";DATABASE=myDatabase;* – replacing mySystem with your system name, myUserID with your user ID on that system, and myPassword with that user ID's password. Note that the string must be specified exactly as shown. Alternatively you can specify just *SYSTEM=mySystem;*

#### Cursor behavior notes:

Cursor behaviors can affect how data is fetched when working with the IBM i Access ODBC driver.

Cursor types can be set via SQLSetStmtAttr with the SQL\_ATTR\_CURSOR\_TYPE option.

**Cursor types:**

- `SQL_CURSOR_FORWARD_ONLY` - All catalog result sets use this type of cursor. When a catalog result set has been generated the cursor type will be automatically changed to this.
- `SQL_CURSOR_KEYSET_DRIVEN` - mapped to `SQL_CURSOR_STATIC`.
- `SQL_CURSOR_DYNAMIC` - supported.
- `SQL_CURSOR_STATIC` -supported if the statement allows it.

**Note:** Procedure result set cursors are opened in the procedure, therefore setting the cursor type with `SQLSetStmtAttr` will not affect the cursor type. See Stored procedure result sets for more information on procedure result sets.

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. 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` on a multiple row fetch is not supported.

**Result set row counts:**

There are several options that your application can use to determine the row count before fetching data:

- You can set the cursor type to `SQL_CURSOR_STATIC`.
- If your application uses ADO, you can use client-side cursors.
- Your application can use the `COUNT()` function by calling `SELECT COUNT(*) FROM MYTABLE` prior to running the actual query.

**Extended dynamic disabled error:**

The IBM i Access ODBC driver displays the *Extended dynamic support disabled* message when a SQL package is unusable. To correct the problem :

1. Delete the SQL package on the system so that when you run your application the package will be created with your default package settings
2. Change the *SQL default library* connection string setting to match the setting that is saved with the SQL package
3. Switch the *Return code for unusable package* ODBC DSN setting to *Ignore* or *Warning*. Alternatively, you can get this same behavior by setting the `PKG` connection string setting.

4. Disable the XDYNAMIC connection string setting.

#### ODBC 64-bit Windows and Linux Considerations:

Identify header files and data types when using the ODBC driver in an IBM i Access for Windows or IBM i Access for Linux environment.

The IBM i Access ODBC driver implements support for 64-bit ODBC APIs. In general, the support is implemented as defined in the ODBC header files provided by Microsoft for Windows environments and unixODBC for Linux environments. When writing code to call the ODBC APIs, refer to the appropriate ODBC header file for the function prototype. The header files are listed below.

- sql.h
- sqlext.h
- sqltypes.h
- sqlucode.h

There is one case, `SQLExtendedFetch`, that is handled differently in Linux than what is defined in `sqlext.h`. In `sqlext.h`, the definition for the `pcrow` parameter is defined as a `SQLROWSETSIZE` pointer. `SQLROWSETSIZE` is a 4 byte value in 64-bit Linux implementations. Despite this, the ODBC driver returns data for the `pcrow` pointer as an 8 byte (64-bit) value to be consistent with its 64-bit Windows ODBC implementation.

Some inherent differences between Windows, Linux, and 64-bit are identified below.

- In 64-bit Linux environments, the size of a long C/C++ type is 8 bytes. In other environments, including 64-bit Windows, the size of the long type is 4 bytes. Refer to the table below.
- In 32-bit environments, the size of a pointer is 4 bytes. In 64-bit environments, the size of a pointer is 8 bytes.
- Some of the ODBC APIs have parameters that are pointers. In some cases, these pointers are used to pass data of different sizes between the application and the driver. In 64-bit implementations, there are some changes where the size of data passed this way has changed from being 4 byte values to 8 byte values.

Some common C/C++ types and the size of each are listed in the table below.

*Table 17. Common C/C++ types and the size of each*

C/C++ Type	Linux 64-bit	Windows 64-bit	Linux 32-bit	Windows 32-bit
int	4	4	4	4
long	8	4	4	4
long long	8	Not defined	8	Not defined
LONG LONG	Not defined	8	Not defined	Not defined
pointer size	8	8	4	4
INT32	Not defined 4 <sup>1</sup>	4	Not defined 4 <sup>1</sup>	4
INT64	Not defined 8 <sup>1</sup>	8	Not defined 8 <sup>1</sup>	8
SQLSMALLINT	2	2	2	2
SQLINTEGER	4	4	4	4
SQLLEN	8	8	4	4
SQLSETPOSIROW	8	8	2	2
SQLROWCOUNT	8	Not defined	4	4
SQLROWSETSIZE	4	Not defined	4	4



Table 17. Common C/C++ types and the size of each (continued)

C/C++ Type	Linux 64-bit	Windows 64-bit	Linux 32-bit	Windows 32-bit
SQLROWOFFSET	8	Not defined	4	4
SQLPOINTER	8	8	4	4
UINT_PTR	Not defined 8 <sup>1</sup>	8	Not defined 4 <sup>1</sup>	4
DWORD	Not defined 4 <sup>1</sup>	4	Not defined 4 <sup>1</sup>	4
SDWORD	Not defined 4 <sup>1</sup>	4	Not defined 4 <sup>1</sup>	4
ULONG_PTR	Not defined 8 <sup>1</sup>	8	Not defined 4 <sup>1</sup>	4
SQLHANDLE	8	8	4	4
SQLHDESC	8	8	4	4

**Note:** 1. This type is not defined in a standard header file. It is defined in the toolkit provided with the IBM i Access for Linux product.

Options for the ODBC APIs listed below result in different behavior by the 32-bit and 64-bit ODBC drivers for parameter pointer data. Generally, the 64-bit ODBC drivers treat parameter pointer data as 8 byte (64-bit) values, unless otherwise noted.

**SQLColAttribute**

SQL\_DESC\_DISPLAY\_SIZE  
 SQL\_DESC\_LENGTH  
 SQL\_DESC\_OCTET\_LENGTH  
 SQL\_DESC\_COUNT

**SQLColAttributes**

SQL\_COLUMN\_DISPLAY\_SIZE  
 SQL\_COLUMN\_LENGTH  
 SQL\_COLUMN\_COUNT

**SQLGetConnectAttr**

SQL\_ATTR\_QUIET\_MODE

**SQLGetConnectOption (This is mapped to SQLGetConnectAttr by the ODBC driver manager.)**

SQL\_ATTR\_QUIET\_MODE

**SQLGetDescField**

SQL\_DESC\_ARRAY\_SIZE

**SQLGetDiagField**

SQL\_DIAG\_CURSOR\_ROW\_COUNT  
 SQL\_DIAG\_ROW\_COUNT  
 SQL\_DIAG\_ROW\_NUMBER

**SQLGetInfo (These are all handled by the ODBC driver manager.)**

SQL\_DRIVER\_HENV  
 SQL\_DRIVER\_HDBC  
 SQL\_DRIVER\_HLIB  
 SQL\_DRIVER\_HSTMT  
 SQL\_DRIVER\_HDESC

**SQLGetStmtAttr**

SQL\_ATTR\_APP\_PARAM\_DESC  
SQL\_ATTR\_APP\_ROW\_DESC  
SQL\_ATTR\_IMP\_PARAM\_DESC  
SQL\_ATTR\_IMP\_ROW\_DESC  
SQL\_ATTR\_MAX\_LENGTH  
SQL\_ATTR\_MAX\_ROWS  
SQL\_ATTR\_PARAM\_BIND\_OFFSET\_PTR  
SQL\_ATTR\_ROW\_ARRAY\_SIZE  
SQL\_ATTR\_ROW\_BIND\_OFFSET\_PTR  
SQL\_ATTR\_ROW\_NUMBER  
SQL\_ATTR\_ROWS\_FETCHED\_PTR  
SQL\_ATTR\_KEYSET\_SIZE

**SQLGetStmtOption (This is mapped to SQLGetStmtAttr by the ODBC driver manager.)**

SQL\_MAX\_LENGTH  
SQL\_MAX\_ROWS  
SQL\_ROWSET\_SIZE  
SQL\_KEYSET\_SIZE

**SQLSetConnectAttr**

SQL\_ATTR\_QUIET\_MODE

**SQLSetConnectOption (This is mapped to SQLSetConnectAttr by the ODBC driver manager.)**

SQL\_ATTR\_QUIET\_MODE

**SQLSetDescField**

SQL\_DESC\_ARRAY\_SIZE

**SQLSetStmtAttr**

SQL\_ATTR\_APP\_PARAM\_DESC  
SQL\_ATTR\_APP\_ROW\_DESC  
SQL\_ATTR\_IMP\_PARAM\_DESC  
SQL\_ATTR\_IMP\_ROW\_DESC  
SQL\_ATTR\_MAX\_LENGTH  
SQL\_ATTR\_MAX\_ROWS  
SQL\_ATTR\_PARAM\_BIND\_OFFSET\_PTR  
SQL\_ATTR\_ROW\_ARRAY\_SIZE  
SQL\_ATTR\_ROW\_BIND\_OFFSET\_PTR  
SQL\_ATTR\_ROW\_NUMBER  
SQL\_ATTR\_ROWS\_FETCHED\_PTR  
SQL\_ATTR\_KEYSET\_SIZE

**SQLSetConnectAttr**

SQL\_MAX\_LENGTH  
SQL\_MAX\_ROWS

SQL\_ROWSET\_SIZE

SQL\_KEYSET\_SIZE

### Restrictions of the 64-bit IBM i Access for Windows ODBC Driver:

MTS is not supported on the 64-bit IBM i Access for Windows ODBC driver.

For more information on MTS see Using distributed transaction support .

### SQLTables Description:

There are multiple considerations when using IBM i Access ODBC driver SQLTables API.

- 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 "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 IBM i operating system does not 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.

### Handle long-running queries:

There are a number of ways to limit the amount of time a query runs with the IBM i Access ODBC driver. Listed below are a couple of options which can be enabled in ODBC.

1. An application can set the SQL\_ATTR\_QUERY\_TIMEOUT connection attribute to specify the maximum amount of time a query can run. Note, the query will not start if the SQL Optimizer determines that the amount of time needed to process the query will exceed the SQL\_ATTR\_QUERY\_TIMEOUT value. If the estimated amount of time exceeds the value of the SQL\_ATTR\_QUERY\_TIMEOUT attribute, an SQL0666 SQLCODE will be returned to the application. The default value for SQL\_ATTR\_QUERY\_TIMEOUT is 0 which indicates that the query will run until completion.
2. An application can call the SQLCancel API. To do this an application needs to be multi-threaded. While the long running query is running on one thread, another thread calls SQLCancel using the same statement handle.

## Isolation level considerations:

Run IBM i Access ODBC autocommit support to different isolation (commit) levels.

IBM i allows you to run ODBC autocommit support to use other isolation levels than just \*NONE.

By specifying an isolation level something other than \*NONE, you can run autocommit under a different isolation level. Be aware that an autocommit commitment levels other than \*NONE require that you make additional other changes and that it changes the behavior of some functions, like eliminating the ability to update non-journaled files. For more information, see the Isolation level topic in the SQL Reference.

There is a SQLDriverConnect keyword called TRUEAUTOCOMMIT which allows an application to control whether or not it run autocommit under the \*NONE isolation level or the SQL\_ATTR\_TXN\_ISOLATION setting. If TRUEAUTOCOMMIT is set to 1 in the SQLDriverConnect connection string then the application will run autocommit using the SQL\_ATTR\_TXN\_ISOLATION setting. If TRUEAUTOCOMMIT is not set, the default value of 0 is used. The default behavior will run autocommit using the \*NONE isolation level.

### Related information

SQL Reference Isolation level

## IBM i Access for Windows ODBC performance

See any of the following IBM i Access ODBC performance topics.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

### Performance-tuning IBM i Access for Windows ODBC:

A key consideration for DB2 for i Access 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:*

The performance characteristics of any computing environment are 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.

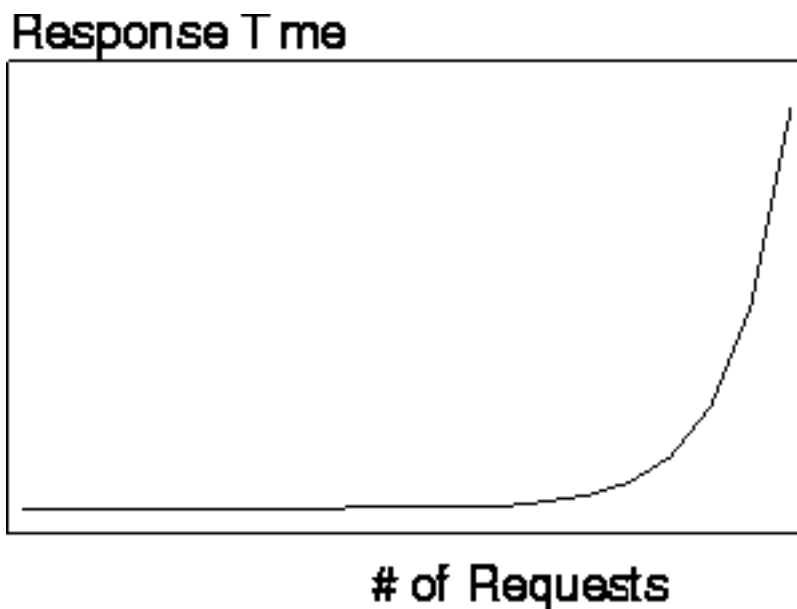
In many servers, capacity is considerable, and is not an issue with users. In others, it is the primary performance concern. Response time is critical. One of the most important questions for administrators is: *How much can the server be degraded (by adding users, increasing utilization) before users 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.

*Performance architecture of the IBM i Access for Windows ODBC driver:*

For the IBM i Access ODBC driver, all of the internal data flows between the client and the server are chained together, and transmitted 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 configurable on the IBM i Access 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.

*Select a stringent level of commitment control:*

There are some important considerations when choosing to use IBM i Access ODBC 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
```

In the V6R1 release, DB2 for i added 'ROW CHANGE TIMESTAMP' support. This allows an application to create a table with a ROW CHANGE TIMESTAMP column, which can be used together with a ROW CHANGE expression and the RID function to guarantee row uniqueness without the need to cache all the columns for each of the rows that might be subsequently changed. This is a better solution for optimistic locking, in that it does not require row locks to be maintained and it does not require caching of every column value in the row to be updated. See the following example:

```
/*Add a row change timestamp column (called 'RCT' in this example)*/
/* to the table when it is created */
CREATE TABLE TABLEX (col1 int,..., RCT GENERATED ALWAYS FOR
EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP NOT NULL);
:
/*Add the ROW CHANGE TOKEN expression and the RID function to the */
/* select list of the query (note : a ROW CHANGE expression which */
/*specifies the TIMESTAMP or the column itself can also be used */
/*in the query. See the SQL Reference for more details). */
/* Note that locks on the rows read by the query do not need to be*/
/* held. */
SELECT ROW CHANGE TOKEN FOR tablex,RID(tablex),col1,...,
FROM TABLEX WHERE ...
:
/* For each row, cache away just the value from the ROW CHANGE */
/* TOKEN and the value for the result of the RID function. */
/* When a row qualifies to be updated, just the ROW CHANGE */
/* TOKEN value and the RID()function value need to be specified in*/
/* the criteria for the UPDATE. */
UPDATE table SET Col1=new_val1, Col2=new_val2,... WHERE ROW CHANGE
TOKEN for tablex = <saved value> and RID(tablex) = <saved RID value>
:
```

If the UPDATE statement returns a 'row not found' error, this indicates that the row you attempted to update has been updated or deleted since the time it was read. See the SQL Reference for more details on ROW CHANGE expression and the RID() function.

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.

## Related information

Commitment control  
DB2 for i SQL Reference

*Fine-tune record-blocking:*

**Record-blocking** is a technique that significantly reduces the number of network flows and therefore improves performance when using IBM i Access ODBC driver.

It does this by returning a *block* of multiple 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 531. 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.

## Related reference

"Connection string keywords - Performance properties" on page 503

Use these IBM i Access ODBC driver connection string keywords to change Performance properties of the ODBC connection.

*Use Extended Dynamic SQL:*

Use the IBM i extended dynamic capability to improve performance of your ODBC applications.

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 is referred to as **static SQL**. One performance advantage to this approach is that SQL statements were optimized at the time the high-level program was compiled, 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 is always advantageous. This works well with a unique feature of DB2 for i. 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 for i automatically regenerates the access plan when it determines that sufficient database changes require reoptimization.

For more information on packages and the types of SQL statements stored in them, see the SQL packages topic in the IBM i Information Center.

### **Related information**

SQL packages

### **Performance considerations of common end-user tools:**

Several tools can help tune your IBM i Access ODBC driver environment.

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<sup>®</sup> Impromptu<sup>®</sup>
- Gupta SQL Windows
- IBM Visualizer for Windows
- Lotus<sup>®</sup> Approach<sup>®</sup>
- Lotus Notes
- Notes<sup>®</sup> Pump
- Microsoft Access
- Microsoft Internet Information Server
- Microsoft SQL Server
- Microsoft Visual Basic
- Powersoft PowerBuilder
- Microsoft Visual Studio .NET

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: Common tool behaviors that degrade ODBC performance:*

The following examples demonstrate performance problems that are associated with writing SQL and IBM i Access ODBC calls that do NOT take advantage of a unique feature of a particular ODBC driver or the server database management system.

*Example: Query tool A:*

This example illustrates using IBM i Access ODBC bound columns to retrieve information faster.

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:*

This example illustrates using one allocation statement for the entire IBM i Access ODBC call.

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. 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 IBM i 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:*

In this example, the complex decision support-type queries ended up making the IBM i Access ODBC query run longer.

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. In the IBM i Access ODBC environment, 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:*

Get answers to the when, what, and how questions when designing your ODBC environment.

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:**

- SQL Tables with over 10,000 rows - Performance impact: **noticeable**

- SQL Tables 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 for i SQL Reference topic in the IBM i Information Center.

**Related information**

DB2 for i SQL Reference

*Database design:*

Use the following topics to determine what tables you require in your DB2 for IBM i database and to understand the relationship between those tables.

*Normalization:*

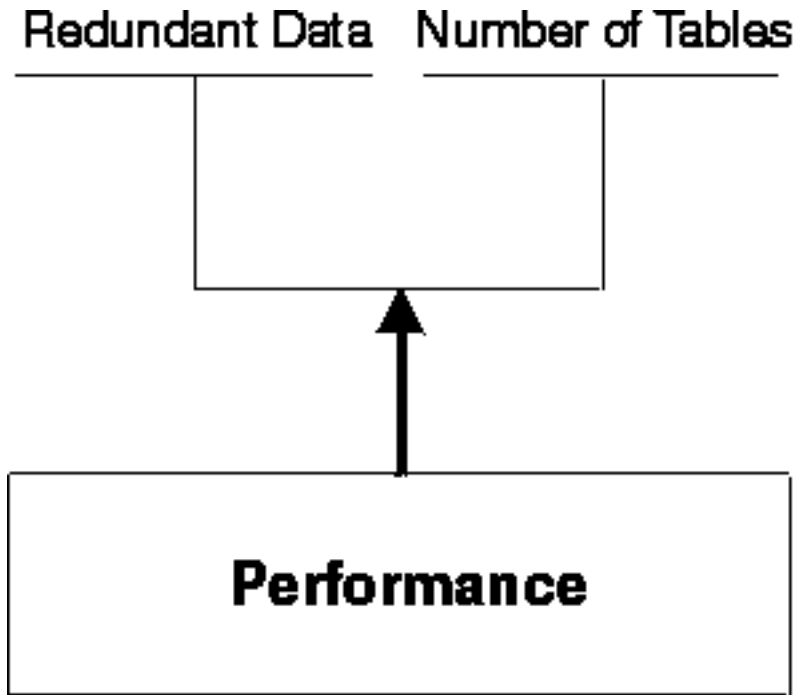
Normalization should be considered when designing DB2 for i database tables and schemas.

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:



*Figure 1. Balancing redundant data and number of tables*

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:

**EMPLOYEE Table**

Employee No	Jobcode
00010	057
00020	054
00030	057
.	.

**JOB CODE Table**

Jobcode	Job Title
054	Programmer
057	Engineer
.	.
.	.

*Figure 2. Normalized data form*

## EMPLOYEE Table

Employee No	Job Title
00010	Engineer
00020	Programmer
00030	Engineer
.	.

Figure 3. Redundant data form

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.

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 ODBC 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 and the queries that access the table always specify a WHERE clause, create an index over the columns that are used in the WHERE clause. The index will allow the DB2 for i optimizer to use the index to access the table. The use of indexes is very important for achieving the best possible performance.

## Related reference

“Optimizer” on page 542

The optimizer is an important part of the DB2 for i database engine because it makes the key decisions for good database performance. Its main objective is to find the most efficient access path to the DB2 for i data.

“Use indexes”

The use of indexes can improve significantly the performance of your IBM i Access ODBC applications.

*Use indexes:*

The use of indexes can improve significantly the performance of your IBM i Access ODBC applications.

The DB2 for i query optimizer uses indexes for performance optimization, and in some cases, is able to read all necessary data to satisfy a query from an index. See the related link for more information on the Optimizer.

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, DELETE, as well as these statements in a MERGE statement, 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.

## Related reference

### “Optimizer”

The optimizer is an important part of the DB2 for i database engine because it makes the key decisions for good database performance. Its main objective is to find the most efficient access path to the DB2 for i data.

### “Table size” on page 540

The size of the tables that your application program accesses has a significant impact on the performance of the ODBC application program.

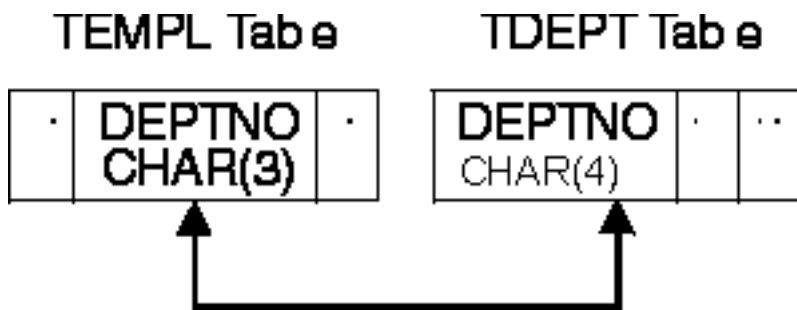
### Match attributes of join fields:

For DB2 for i, columns used to join tables should have the same attributes.

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:

The optimizer is an important part of the DB2 for i database engine because it makes the key decisions for good database performance. Its main objective is to find the most efficient access path to the DB2 for i 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, the optimizer does the following:

- Determines possible implementations
- Picks the optimal implementation for running of the SQL statement



## Related reference

“Use indexes” on page 541

The use of indexes can improve significantly the performance of your IBM i Access ODBC applications.

“Table size” on page 540

The size of the tables that your application program accesses has a significant impact on the performance of the ODBC application program.

### *Cost estimation:*

At runtime, the DB2 for i optimizer chooses an optimal access method for the query by calculating an implementation cost based on the current state of the tables referenced in the query and any access paths (indexes) which are available.

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 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, the DB2 for i optimizer uses a general set of guidelines to choose the best method for accessing the database tables.

The 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, the 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 optimizer spends little time in query optimization. For large tables, the optimizer considers more indexes. Generally, the optimizer considers five or six indexes (for each table of a join) before running out of optimization time.

#### | **ODBC support for multiple row statements:**

| DB2 for IBM i and ODBC supports multiple row operations on INSERT, UPDATE, DELETE, and MERGE  
| statements using the technique described below. This example shows how to use the multiple row  
| INSERT statement in ODBC to insert multiple rows into a DB2 for i table.

| The multiple row **INSERT** statement provides a means to insert multiple rows with a single **SQLExecute**  
| request. From a performance standpoint, it provides the best way 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
- | • multiple row 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 multiple row INSERT statement most efficiently performs inserts into a table when multiple rows can  
| be cached on the client and sent at once. The advantages with multiple row 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 multiple row INSERT statements.

| Example:

```
| INSERT INTO TEST.TABLE1 ? ROWS VALUES (?, ?, ?, ?)
```

| The INSERT statement has additional syntax that identifies it as a multiple row INSERT. This optional  
| syntax, 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 that this extra clause is not needed; a multiple row INSERT statement can  
| also run by preparing a INSERT with VALUES form of the statement with parameter markers, setting the  
| row count on the **SQLSetStmtAttr** API, and then executing the statement.

| To view examples of the APIs used from a C program for multiple row statements , see the multiple row  
| insert and multiple row fetch C example topic.

## | **Related reference**

- | “Multiple row INSERT and multiple row FETCH examples” on page 476
- | Multiple row inserts and multiple row fetches can be used to enhance the performance of an IBM i Access ODBC application.

## **Catalog functions:**

Catalog functions return information about the DB2 for i database objects with which you are working.

To process ODBC **SQLTables** requests, logical files are built over the server cross reference file QADBXRREF in library QSYS. QADBXRREF 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, including SQL SQL TABLES and VIEWS.

### **TABLE**

Selects all PHYSICAL files, including SQL TABLES that are not server files (cross reference or data dictionary).

**VIEW** Selects all LOGICAL files, including SQL VIEWS that are not server files (cross reference or data dictionary).

### **SYSTEM TABLE**

Selects all PHYSICAL and LOGICAL files, including SQL VIEWS that are either server files or data dictionary files.

### **TABLE, VIEW**

Selects all LOGICAL and PHYSICAL files, including 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 system 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:**

The Appendix of the *SQL Reference* book contains additional information. View an HTML online version of the book, or print a PDF version, from the DB2 for i SQL Reference topic in the IBM i Information Center.

## **Related information**

DB2 for i SQL Reference

## **Exit programs:**

There are requirements when calling an IBM i Access ODBC exit program.

An exit program is a program to which control is passed from a calling program. 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 five 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\_SQL2**

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.

*Examples: User exit programs:*

The following examples do not show all of the programming considerations or techniques. Review the examples before you begin IBM i Access ODBC application design and coding.

*Example: ILE C/400 user exit program for exit point QIBM\_QZDA\_INIT:*

The following ILE C/400® program handles IBM i Access ODBC security by rejecting requests from certain users. It can be used as a shell for developing exit programs tailored for your operating environment.

```
| /*****
| /*      Sample Exit Program      */
| /*      */
| /*      Exit Point Name      : QIBM_QZDA_INIT      */
| /*      */
| /*      Description      : The following ILE C Language program      */
| /*                          handles ODBC security by rejecting      */
| /*                          requests from users who use ODBC and      */
| /*                          signon using a user profile of 'GUEST'.      */
| /*                          It can be used as a shell program      */
| /*                          for developing exit programs tailored      */
| /*                          for your environment.      */
| /*****
| #include <stdio.h>
| #include <string.h>
| #include <ezdaep.h>      /* ZDA exit program formats */
| main(int argc, char *argv[])
```

```

| {
|   Qzda_Init_Format_t input;          /* input format          */
|
|   /*****
|   /* Copy format parameter to local storage          */
|   /*****
|   memcpy(&input,(Qzda_Init_Format_t *) argv[2],
|           sizeof(Qzda_Init_Format_t));
|
|   /*****
|   /* If user profile is 'GUEST' and interface type is 'ODBC'          */
|   /* reject the connection.          */
|   /*****
|   if (memcmp(input.User_Profile,"GUEST",10)==0 &&
|       memcmp(input.Interface_Type,"ODBC",4) == 0)
|   /*****
|   /* Reject the connection.          */
|   /*****
|   strcpy(argv[1],"0");
|   else
|   /*****
|   /* Allow the connection.          */
|   /*****
|   strcpy(argv[1],"1");
|   return;
| }

```

Example: CL user exit program for exit point QIBM\_QZDA\_INIT:

The following Control Language program handles IBM i Access ODBC security by rejecting requests from certain users. It can be used as a shell for developing exit programs tailored for your operating environment.

```

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*          @ss1s@ 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:*

The following ILE C/400 program will reject any UPDATE request for user GUEST. It can be used as a shell for developing IBM i Access ODBC exit programs tailored for your operating environment.

```

/*-----*/
*           @ss1s@@ 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 @@sql1@@ 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 (! (strncmp(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 @@sql1@@ 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 @@sql1@@ 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 @@sqli@@ 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:*

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 IBM i Access ODBC exit programs tailored for your operating environment.

```

/*-----
*                               @@s1s@@ 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 (*RTVOBJINF) */
    char Exit_format_name[8];   /* User exit format name (ZDAR0100) */
    long Requested_function;    /* function being preformed */
    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("QGGL/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 IBM i database 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
- Return package information
- Stream fetch
- 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\_SQL2 format ZDAQ0200:

The following table shows parameter fields and their descriptions for the IBM i database exit program called at exit point QIBM\_QZDA\_SQL2 with the ZDAQ0200 format.

Table 18. 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.
28	1C	BINARY(4)	Requested function	The function being performed.  This field contains one of the following: <ul style="list-style-type: none"> <li>• X'1800' - Prepare</li> <li>• X'1803' - Prepare and describe</li> <li>• X'1804' - Open/describe</li> <li>• X'1805' - Execute</li> <li>• X'1806' - Execute immediate</li> <li>• X'1809' - Connect</li> <li>• X'180C' - Stream fetch</li> <li>• X'180D' - Prepare and execute</li> <li>• X'180E' - Open and fetch</li> <li>• X'180F' - Create package</li> <li>• X'1810' - Clear package</li> <li>• X'1811' - Delete package</li> <li>• X'1812' - Execute or open</li> <li>• X'1815' - Return package information</li> </ul>
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 dynamic SQL package.
92	5C	BINARY(2)	DRDA indicator	<ul style="list-style-type: none"> <li>• 0 - Connected to local RDB</li> <li>• 1 - Connected to remote RDB</li> </ul>

Table 18. Exit point QIBM\_QZDA\_SQL2 format ZDAQ0200 (continued)

Offset		Type	Field	Description
Dec	Hex			
94	5E	CHAR(1)	Commitment control level	<ul style="list-style-type: none"> <li>'A' - Commit *ALL</li> <li>'C' - Commit *CHANGE</li> <li>'N' - Commit *NONE</li> <li>'S' - Commit *CS (cursor stability)</li> </ul>
95	5F	CHAR(10)	Default SQL collection	<p>Name of the default SQL schema used by the IBM i Database Server. If the actual default SQL schema name is greater than 10 bytes, the following special value will be passed, indicating that the default SQL schema name should be obtained from the 'Extended SQL Schema' field:</p> <ul style="list-style-type: none"> <li>*EXTDSCHMA</li> </ul> <p>Note: The Extended SQL Schema field will always be set, even if length is less than 10. Users can always refer to that field to get the Default SQL Schema name.</p>
105	69	CHAR(3)	Reserved	Reserved for future parameters.
108	6C	BINARY(4)	Offset to the extended cursor name	The offset in this structure to the extended cursor name
112	70	BINARY(4)	Length of the extended cursor name	Length, in bytes, of the extended cursor name
116	74	BINARY(4)	Offset to the Extended SQL Schema	The offset in this structure to the Extended SQL Schema.
120	78	BINARY(4)	Length of the Extended SQL Schema	Length, in bytes, of the Extended SQL Schema.
124	7C	CHAR(110)	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 2 MB (2,097,152 bytes).
238	EE	CHAR(*)	SQL statement text	Entire SQL statement.
*	*	CHAR(*)	Extended Cursor Name	The extended cursor name.
*	*	CHAR(*)	Extended Schema Name	The extended schema name.

**Note:** This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLESRC 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 IBM i database exit program called at exit point QIBM\_QZDA\_INIT using the ZDAI0100 format.

Table 19. 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, QRPGSRC, QRPGLESRC, QCBLSRC and QCBLESRC 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 IBM i database exit program called at exit point QIBM\_QZDA\_NDB1 using the ZDAD0100 format.

Table 20. 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 20. 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: <ul style="list-style-type: none"> <li>• X'1800' - Create source physical file</li> <li>• X'1801' - Create database file, based on existing file</li> <li>• X'1802' - Add database file member</li> <li>• X'1803' - Clear database file member</li> <li>• X'1804' - Delete database file member</li> <li>• X'1805' - Override database file</li> <li>• X'1806' - Delete database file override</li> <li>• X'1807' - Create save file</li> <li>• X'1808' - Clear save file</li> <li>• X'1809' - Delete file</li> </ul>
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
<p><b>Note:</b> This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLSRC, QCBLSRC and QCBLESRC in library QSYSINC.</p>				

Parameter fields for exit point QIBM\_QZDA\_NDB1 format ZDAD0200:

The following table shows parameter fields and their descriptions for the IBM i database exit program called at exit point QIBM\_QZDA\_NDB1 by using the ZDAD0200 format.

Table 21. 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 21. 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 QCBLESRC 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 IBM i database exit program called at exit point QIBM\_QZDA\_SQL1 using the ZDAQ0100 format.

Table 22. 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 22. 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: <ul style="list-style-type: none"> <li>• X'1800' - Prepare</li> <li>• X'1803' - Prepare and describe</li> <li>• X'1804' - Open/Describe</li> <li>• X'1805' - Execute</li> <li>• X'1806' - Execute immediate</li> <li>• X'1809' - Connect</li> <li>• X'180D' - Prepare and execute or prepare and open</li> <li>• X'180E' - Open and fetch</li> <li>• X'180F' - Create package</li> <li>• X'1810' - Clear package</li> <li>• X'1811' - Delete package</li> <li>• X'1812' - Execute or open</li> <li>• X'1815' - Return package information</li> </ul>
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	<ul style="list-style-type: none"> <li>• 0 - Connected to local RDB</li> <li>• 1 - Connected to remote RDB</li> </ul>
94	5E	CHAR(1)	Commitment control level	<ul style="list-style-type: none"> <li>• 'A' - Commit *ALL</li> <li>• 'C' - Commit *CHANGE</li> <li>• 'N' - Commit *NONE</li> <li>• 'S' - Commit *CS (cursor stability)</li> </ul>
95	5F	CHAR(512)	First 512 bytes of the SQL statement text	First 512 bytes of the SQL statement

**Note:** This format is defined by member EZDAEP in files H, QRPGSRC, QRPGLSRC, QCBLSRC and QCBLESRC in library QSYSINC.

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 IBM i database exit program called at exit point QIBM\_QZDA\_ROI1 using the ZDAR0100 format.

*Table 23. 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	<p>The function being performed.</p> <p>This field contains one of the following:</p> <ul style="list-style-type: none"> <li>• X'1800' - Retrieve library information</li> <li>• X'1801' - Retrieve relational database information</li> <li>• X'1802' - Retrieve SQL package information</li> <li>• X'1803' - Retrieve SQL package statement information</li> <li>• X'1804' - Retrieve file information</li> <li>• X'1805' - Retrieve file member information</li> <li>• X'1806' - Retrieve record format information</li> <li>• X'1807' - Retrieve field information</li> <li>• X'1808' - Retrieve index information</li> <li>• X'180B' - Retrieve special column information</li> </ul>

Table 23. Exit point QIBM\_QZDA\_ROI1 format ZDAR0100 (continued)

Offset		Type	Field	Description
Dec	Hex			
32	20	CHAR(20)	Schema name	The Schema or search pattern used when retrieving information about schemas, packages, package statements, files, members, record formats, fields, indexes, and special columns. If schema name length or search pattern length is greater than 20, the following special value will be passed, indicating that the schema name should be obtained from the 'Extended Schema name' field: <ul style="list-style-type: none"> <li>*EXTDSCHMA</li> </ul> Note: The Extended schema name field will always be set, even if length is less than 20. Users can always refer to that field to get the schema name.
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.
404	194	CHAR(256)	Extended Schema Name	Extended Schema name or search pattern used.
<b>Note:</b> This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLRSC 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 IBM i database exit program called at exit point QIBM\_QZDA\_ROI1 using the ZDAR0200 format.

Table 24. 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.

Table 24. Exit point QIBM\_QZDA\_ROI1 format ZDAR0200 (continued)

Offset		Type	Field	Description
Dec	Hex			
28	1C	BINARY(4)	Requested function	The function being performed.  This field contains one of the following: <ul style="list-style-type: none"> <li>• X'1809' - Retrieve foreign key information</li> <li>• X'180A' - Retrieve primary key information</li> </ul>
32	20	CHAR(10)	Primary key table schema name	The name of the schema that contains the primary key table used when retrieving primary and foreign key information. When the name is greater than 10 bytes, the following special value will be passed, indicating that the primary key table schema name should be obtained from the 'primary key table extended schema name' field: <ul style="list-style-type: none"> <li>• *EXTDSCHMA</li> </ul> <p>Note: The 'Primary key table extended schema name' field will always be set, even if length is less than 10. Users can always refer to that field to get the schema name.</p>
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 schema name	The name of the schema that contains the foreign key table used when retrieving foreign key information. When the name is greater than 10 bytes, the following special value will be passed, indicating that the foreign key table schema name should be obtained from the 'foreign key table extended schema name' field: <ul style="list-style-type: none"> <li>• *EXTDSCHMA</li> </ul> <p>Note: The 'Foreign key table extended schema name' field will always be set, even if length is less than 10. Users can always refer to that field to get the schema name.</p>
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.
308	134	CHAR(128)	Primary key table extended schema name	The name of the schema that contains the primary key table used when retrieving primary key information
436	1B4	CHAR(128)	Foreign key table extended schema name	The name of the schema that contains the foreign key table used when retrieving foreign key information
<p><b>Note:</b> This format is defined by member EZDAEP in files H, QRPGRSRC, QRPGLSRC, QCBLSRC and QCBLLESRC in library QSYSINC.</p>				

### SQL and External procedures:

SQL and external procedures are supported on IBM i for database access.

Procedures are, in general, any program that can be executed using an SQL CALL statement. They are commonly used in client/server applications, especially in the area of online transaction processing (OLTP), since they can provide performance, transaction-integrity and security benefits. In DB2 for i , procedures can be written in SQL procedure language or in a number of external programming languages, such as ILE RPG or ILE COBOL. For information regarding specific SQL statements that are used in the examples of these procedures, see the DB2 for i SQL Reference topic collection in the IBM i Information Center.

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).

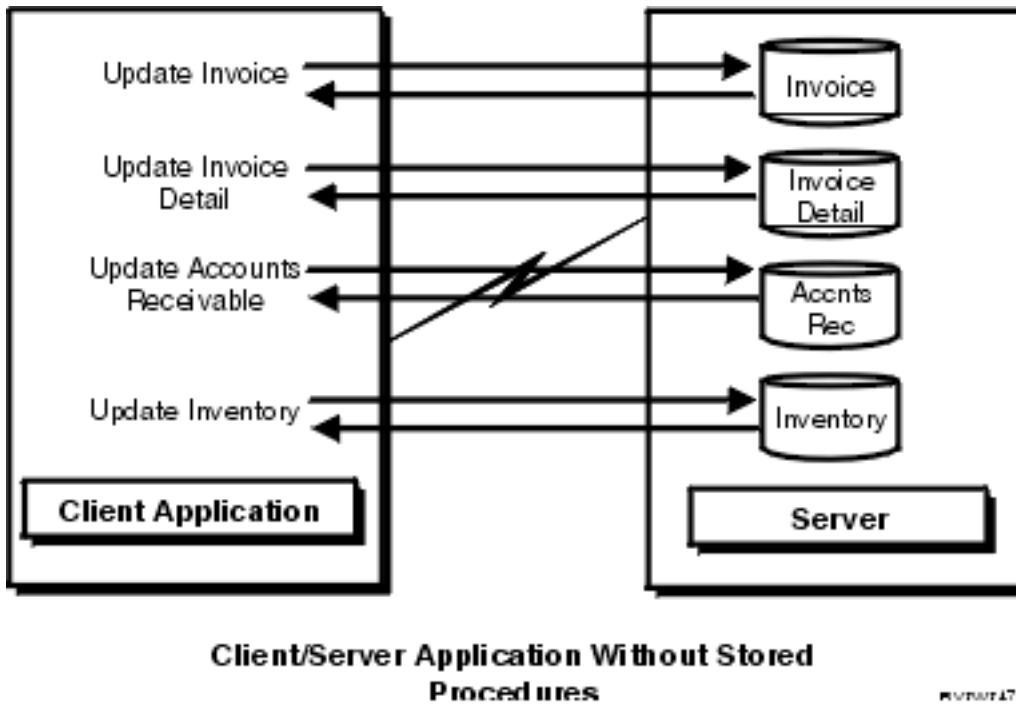
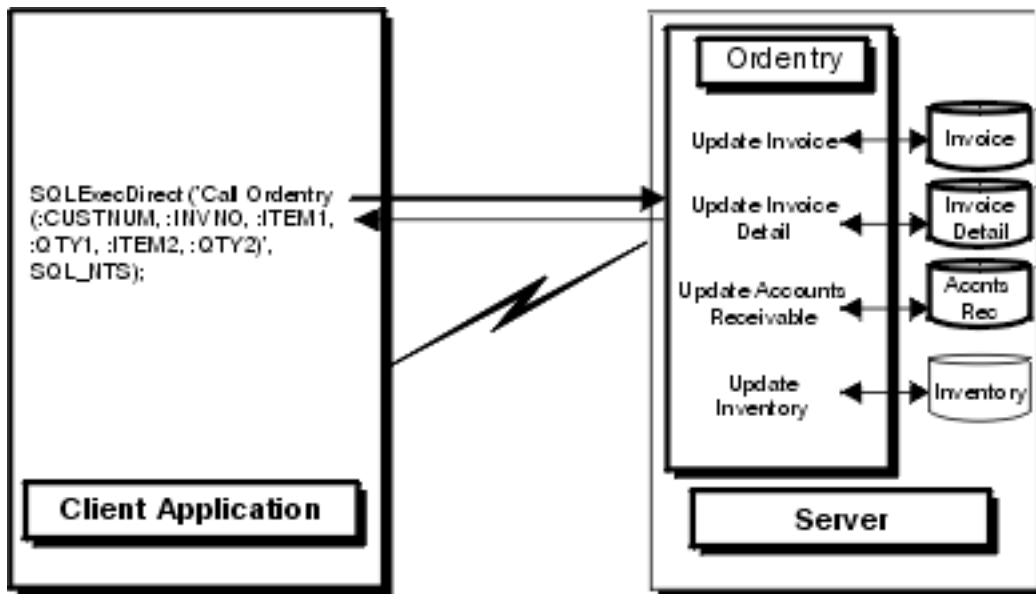


Figure 4. Client/server application without stored procedure

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 i stored procedure can be any IBM i program, and does not have to use SQL for data access.



**Client/Server Application With Stored Procedure**

Figure 5. Client/server application with stored procedure

#### Related tasks

“Call stored procedures” on page 475

Use stored procedures to improve the performance and function of an IBM i Access ODBC application.

#### Related reference

“ODBC program examples” on page 571

ODBC programming examples demonstrate IBM i Access queries and stored procedures.

#### Related information

DB2 for i SQL Reference

#### Procedure result sets:

You can scroll IBM i SQL procedure result sets.

An application can have scrollable result sets returned from a procedure executed using an SQL CALL statement. To take advantage of this support, make the following two changes.

1. Create the procedure with the cursor defined as scrollable.
  - a. This is done by adding the **SCROLL** keyword into the cursor declaration inside the procedure definition. In the following two examples, the stored procedure returns a scrollable result set while the second one does not.
    - CREATE PROCEDURE MYLIB.SCROLLSP ( ) RESULT SETS 1 LANGUAGE SQL  
 sqlproc: begin  
 DECLARE CUR1 SCROLL CURSOR FOR  
 SELECT \* FROM QIWS.QCUSTCDT;  
 OPEN CUR1;  
 SET RESULT SETS CURSOR CUR1;  
 end
    - CREATE PROCEDURE MYLIB.NOSCROLLSP ( ) RESULT SETS 1 LANGUAGE SQL  
 sqlproc: begin  
 DECLARE CUR1 CURSOR FOR

```

SELECT * FROM QIWS.QCUSTCDT;
OPEN CUR1;
SET RESULT SETS CURSOR CUR1;
end

```

2. Code the application using ODBC to ask for a scrollable cursor type.

- a. Call the SQLSetStmtAttr API.
- b. Set the SQL\_ATTR\_CURSOR\_TYPE option to SQL\_CURSOR\_DYNAMIC.

If an attempt is made to scroll backwards with a procedure that did not specify a scrollable cursor, several different problems can occur. In most cases an error is returned from the server indicating scrolling is invalid, and in some cases incorrect data is returned.

Even if the procedure returns multiple result sets, you can only use one cursor type. ODBC either returns an error or ignores the cursor type when a different cursor type is specified for the second result set. To use a scrollable result set as one of the result sets, the application needs to set the cursor type to be scrollable as defined above.

Any attempts to use a result set cursor as an updateable cursor will return an error or be ignored. Procedure result sets are read-only.

Cursor sensitivity may not be honored with procedure result sets, since the cursor was opened when the procedure was run. Cursor sensitivity is controlled by the way the cursor is defined when creating the procedure.

*Examples: Stored procedures:*

View examples of DB2 for IBM i procedures.

*Example: Run CL commands that use SQL stored procedures and ODBC:*

Stored procedure support provides a means to run IBM i 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
- Doing other environmental setup for an application

The following examples show cases where an IBM i CL command is run by using the CALL statement which calls the program designed for running CL commands from SQL. That program (QCMDXEC in library QSYS2) expects two parameters:

1. A string that contains the command text to execute
2. An integer that contains the length of the command text

The parameters must include these attributes for the command to be interpreted properly.

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(JMBLIB/TESTER) MBR(N02) 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 QSYS2.QCMDEXC('OVRDBF FILE(TESTER) TOFILE(MYLIB/");
strcat(stmt, "TESTER) MBR(NO2) OVRSCOPE(*JOB)', 64)");
rc = SQLExecDirect(hstmt, stmt, SQL_NTS);
```

Statements now run against file MYLIB/TESTER will reference member NO2 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.

### Related concepts

“Implementation issues of ODBC APIs” on page 490

Learn about implementations issues when using IBM i Access ODBC APIs.

*Example: Stored procedure calls from Visual Basic with return values:*

The following example of Visual Basic source code shows how to call an DB2 for IBM i procedure and then retrieve the returned values into Visual Basic variables.

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 a DB2 for IBM i procedure and return result values. See “Code directly to ODBC APIs” on page 474 for more information.

```
'*****
'*
'* 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 an IBM i procedure. 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 system.          *
'* 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 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 system.
' *
'*****

```

```

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

```

*Example: Call an IBM i stored procedure by using Visual Basic:*

The Visual Basic programming examples listed below show an IBM i procedure call being prepared.

Two statements are shown:

1. A statement for the creation of the procedure
2. A statement to prepare the call

Create the procedure only once. The definition that it provides is available to ODBC applications, and any other application that can run SQL statements.

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: Call CL command using SQL CALL statement:*

It is possible to run IBM i commands by using an SQL CALL statement. The two examples that are provided here apply to ODBC programs.

Simply call Execute Command (QCMDEXC) to run the command. The process is easy, simply provide the command string and the length of the command string as parameters on the CALL statement. Use the Remote Command API as an alternative.

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 server job).

The second example allows a member other than the first of a multi-member file to be accessed using SQL. You cannot create a multi-member file through CREATE TABLE. However, the following example shows you how to access a member other than the first of a multi-member 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 qsys2.qcmdexc('STRDBG UPDPROD(*YES)',20)"

' Put the system 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)',68)"

' Override the IBM i 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
```

*Tips: Run and call IBM i procedures:*

Use these tips for running and calling DB2 for IBM i procedures.

### Running an IBM i procedure

ODBC provides a standard interface for calling database procedures. The implementation of database procedures differs significantly across various databases. This simple example follows the recommended approach for running an IBM i procedure.

1. Set up a **CREATE PROCEDURE** statement for the procedure and create it. The creation of the procedure defines the procedure and only needs to be done once. The definition that it provides is available to all applications which run against the database, including ODBC applications.

2. Prepare the **CALL** statement to call the procedure.
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 procedure.

### Calling IBM i 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: Call an IBM i stored procedure by using Visual Basic” on page 568 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 IBM i procedure can affect performance significantly. Whenever possible, avoid closing the program with **exit()** in C language and with **SETON LR** in RPG languages. Preferably, use **RETRN** or **return**, but you may need to re-initialize variables on each call, and by-pass file opens.

### ODBC program examples

ODBC programming examples demonstrate IBM i Access queries and stored procedures.

The IBM i Access ODBC programming examples listed below demonstrate how to perform simple queries and how to call stored procedures to access and return data. C/C++, Visual Basic, and RPG programming language versions are provided.

Many of the C/C++ samples are not complete programs. For complete discussions and programming samples, review the following information:

- To access ODBC programming samples for Visual Basic, C++, and Lotus Script programming environments, select the related link below to the IBM ftp site on the Web. Select **index.txt** to see what programming examples are available and to download them to your PC.
- See the Stored procedures topic collection in the IBM i Information Center for information on stored procedures and examples on how to call them.
- Search for ODBC samples in Microsoft’s MSDN library or ODBC webpage for Visual Basic, ADO, and C/C++ examples.
- Also see the C programming example in the Programmer’s Toolkit.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

#### Related reference

“SQL and External procedures” on page 562

SQL and external procedures are supported on IBM i for database access.

#### Related information

 [IBM ftp site](#)

[MSDN Library](#)

#### Example: Visual C++ - Access and return data by calling a procedure:

This example illustrates using Visual C++ to access and return data by a call to a DB2 for IBM i procedure.

Only the code relevant to the procedure call has been included here. This code assumes the connection has already been established. See the Examples: RPG - Host code for ODBC procedures topic for the source code for the procedure.

### Creating the 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 procedure
// Unless the
// procedure is destroyed, this statement need never be run again
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 to call the procedure

```

// Prepare the procedure call
strcpy(szStoredProc, "call partqry2(?, ?)");
// Prepare the CALL 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 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_dsSPartDate,
    10, &lcbBuffer);

```

### Calling the procedure

```

// Request a single record
m_lOption = ONE_RECORD;
m_lPartNo = PartNo;

```

```

// Run the 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 - Access and return data by a call to a procedure:

A Visual Basic example demonstrates creating, preparing, binding, and calling a DB2 for IBM i procedure.

Visual Basic can call external functions that are found in DLLs. Since all ODBC drivers are DLLs, a Visual Basic application can code directly to ODBC APIs to call a procedure and return result values and result sets. See the Code directly to ODBC APIs topic for more information. See the Examples: RPG - Host code for ODBC procedures topic for the source code for the procedure.

#### Creating the procedure

```

' This statement will drop an existing procedure
szDropProc = "drop procedure apilib.partqry2"

'* This statement is used to create a procedure
'* Unless the
'* procedure is destroyed, this statement need never be run again
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 for calling the procedure

```
'* This statement will be used to call the procedure
szStoredProc = "call partqry2(?, ?)"
'* Prepare the 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 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 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, sSDescription, _
        25, 1CbBuffer)
    'Get Quantity. SQLGetLongData uses alias SQLGetData
    rc = SQLGetLongData(hstmt, 3, SQL_C_SLONG, 1SQuantity, _
        Len(1SQuantity), 1CbBuffer)
    'Get Price. SQLGetDoubleData uses alias SQLGetData
    rc = SQLGetDoubleData(hstmt, 4, SQL_C_DOUBLE, dSPrice, _
        Len(dSPrice), 1CbBuffer)
    'Get Received date
    rc = SQLGetData(hstmt, 5, SQL_C_CHAR, sSReceivedDate, _
        10, 1CbBuffer)
    txtDescription = sSDescription 'Show description
    txtQuantity = 1SQuantity 'Show quantity
    txtPrice = Format(dSPrice, "currency") 'Convert dSPrice to
    txtReceivedDate = CDate(sSReceivedDate) 'Convert string to d
    Call DisplayMessage("Record found")
End If
```



## Related reference

“Code directly to ODBC APIs” on page 474

Many PC applications make ODBC calls that allow the user to seamlessly access data on different platforms. Before you begin developing your own IBM i Access application with ODBC APIs, you should understand how an ODBC application connects to and exchanges information with a database server.

“Examples: ILE RPG - Host code for ODBC procedures”

In this example, the program, **SPROC2**, is called from the client as a procedure via IBM i Access ODBC using a CALL statement. It returns data to the client from the PARTS database file.

## Examples: ILE RPG - Host code for ODBC procedures:

In this example, the program, **SPROC2**, is called from the client as a procedure via IBM i Access ODBC using a CALL statement. It returns data to the client from the PARTS database file.

### 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+  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

```

#### Related reference

“Example: Visual Basic - Access and return data by a call to a procedure” on page 573

A Visual Basic example demonstrates creating, preparing, binding, and calling a DB2 for IBM i procedure.

## IBM i Access database APIs

- | Use other technologies for functions that were provided by the IBM i Access for Windows proprietary
- | C/C++ Database APIs, that are no longer being enhanced.

The IBM i Access for Windows proprietary C/C++ Database APIs provided support for IBM i database and catalog functions, in addition to SQL access to IBM i database files.

See other topic collections for details on the following technologies that continue to provide the functions of these deprecated APIs:

- NET Framework Classes
- ADO/OLE DB
- ODBC
- JDBC
- Database Transfer
- ActiveX automation objects

#### Related reference

“Database APIs return codes” on page 22

There are IBM i Access for Windows database APIs return codes.

---

## Java programming

The IBM Toolbox for Java is shipped with the IBM i Access for Windows product is also used independently.

The **Java** programming language, which was defined by Sun, enables the development of portable Web-based applications.

The IBM Toolbox for Java, which is shipped with the IBM i Access for Windows product, provides Java classes for accessing IBM i resources. IBM Toolbox for Java uses the IBM i Access for Windows IBM i host servers as access points to the system. However, you do not need the IBM i Access for Windows product to use IBM Toolbox for Java. You can use the Toolbox to write applications that run independently of the product.

The IBM Toolbox for Java interface behaviors such as security and tracing can differ from those of other IBM i Access for Windows interfaces.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

---

## ActiveX programming

ActiveX automation is a programming technology that is defined by Microsoft and is supported by the IBM i Access for Windows product.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 578.

IBM i Access for Windows provides the following methods for accessing IBM i resources by using ActiveX automation:

### Automation objects:

These objects provide support for:

- Accessing IBM i data queues
- Calling IBM i application programming interfaces and user programs
- Managing IBM i connections and validating security
- Running IBM i CL commands
- Performing data-type and code-page conversions
- Performing database transfers
- Interfacing with host emulation sessions

### IBM i Access for Windows OLE DB provider:

Call the IBM i Access for Windows OLE DB Provider, by using Microsoft’s ActiveX Data Objects (ADO), to access the following IBM i resources:

- The IBM i database, through record-level access
- The IBM i database, through SQL
- SQL stored procedures
- Data queues
- Programs
- CL commands

### Custom controls:

ActiveX custom controls are provided for:

- IBM i data queues
- IBM i CL commands
- IBM i names for previously connected systems
- IBM i Navigator

### **Programmer's Toolkit:**

For detailed information on IBM i Access for Windows ActiveX, see the **ActiveX** topic in the **Programmer's Toolkit** component of product. 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 *Install the Programmer's Toolkit*).
2. Launch the **Programmer's Toolkit** (see *Launch the Programmer's Toolkit*).
3. Select the **Overview** topic.
4. Select **Programming Technologies**.
5. Select **ActiveX**.

### **Related tasks**

"Install the Programmer's Toolkit" on page 5

The Programmer's Toolkit is installed as a feature of the IBM i Access for Windows product.

"Launch the Programmer's Toolkit" on page 5

The Programmer's Toolkit is launched as a feature of the IBM i Access for Windows product.

### **Related reference**

"IBM i Access for Windows OLE DB provider" on page 469

Supports record-level access and SQL access to IBM i database files. Use the ActiveX Data Objects (ADO) and the OLE DB interfaces to take advantage of this support.

---

## **Code license and disclaimer information**

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.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

---

## Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA  
3605 Highway 52 N  
Rochester, MN 55901  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

- | The licensed program described in this document and all licensed material available for it are provided
- | by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement,
- | IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

- | This information contains sample application programs in source language, which illustrate programming
- | techniques on various operating platforms. You may copy, modify, and distribute these sample programs
- | in any form without payment to IBM, for the purposes of developing, using, marketing or distributing
- | application programs conforming to the application programming interface for the operating platform for
- | which the sample programs are written. These examples have not been thoroughly tested under all
- | conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these
- | programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be
- | liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

- | © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. ©
- | Copyright IBM Corp. \_enter the year or years\_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Programming Interface Information

This IBM i Access for Windows publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

---

## Trademarks

- | IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).
- | Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
- | Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

- | Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

---

## Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

**Personal Use:** You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

**Commercial Use:** You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.







Printed in USA