Orchard Software Corporation

Plugin Documentation

Win32API

Version 6.6.3

Originally Issued – March 2001

Revised – August 2015

Orchard Software Corporation

701 Congressional Blvd. • Suite 360 • Carmel, IN 46032

Phone 800 856-1948 • www.orchardsoft.com

Table of Contents

Overview

Orchard Software Win32API Plugin

The Win32API Plugin developed by Orchard Software Corporation allows you to use a subset of the Win32 API function calls from within the 4D environment. Some functions are not Win32 API calls but use the registry or other system files. After the table of contents, the next two sections in this document provide details on each of the System or GUI methods in the plugin, including a description of what each call does, the parameters used in the call, and the values that the call returns. The final section includes a table that contains the numerical equivalent of the constants used in the System and GUI methods.

Compiling Note

If you want to compile the source code for the Win32API plugin, you must have iphlpAPI.h and iphlpAPI.lib from the July 2001 Microsoft SDK. This library is used for the **sys_GetRoutes** function.

Also note that the 4D Plugin SDK 4DPluginAPI.c source code has been modified to retain some compatibility with 4D versions prior to 6.7. Thanks to Thibaud Arguillere.

Version Changes

New in Version 6.6.3

Bug Fixes

- Fixed a bug that could rarely cause 4D to crash when calling **sys_GetPrintJob** repeatedly.

New in Version 6.6.2

Bug Fixes

- Fixed a memory leak in **PA_SetTextInArray**.
- Fixed a memory leak in **sys_SendRawPrinterData**.
- Fixed a memory leak in **gui_GetOpenFileName**.
- Fixed a memory leak in **CStringToUnistring**.
- Fixed a memory leak in **sys_GetPrintJob**.
- Fixed several memory leaks in **TWAIN_AcquireImage**.
- Fixed a memory leak in **createNewProcess**.
- Fixed a memory leak in **gui_GetSaveFileName**.
- Fixed a memory leak in **gui_LoadBackground**.
- Fixed multiple bugs in **sys_EncryptAES** and **sys_DecryptAES** that could cause issues when executed on 4D server.
- Fixed a bug in **sys_GetPrintJob** that could crash 4D.

Changes

- Removed **sys_DeleteRegKey64** as it causes Win32API to crash on Windows XP.
- Added the command **gui_ServerUnloadBackground**.
- Added the command **gui_TakeScreenshot**.

New in Version 6.6.1

Bug Fixes

- Fixed a memory leak that was present in **sys_ShellExecute**.

Changes

- Added the commands **sys_EncryptAES** and **sys_DecryptAES**, which encrypt/decrypt the given text message.

New in Version 6.6.0

Bug Fixes

- Fixed **sys_GetPrintJob** to correctly return all members of the printer array (excluding PS_SOURCE) when passing "PRINT SETTINGS(2)" as the 4DCommand parameter.

Changes

- Added the command **sys_DeleteRegValue**, which deletes the given value from the given registry key.
- Added the command **sys_DeleteRegKey**, which deletes the given registry key. Note: This will delete the key from only the 32-bit view on 64-bit Windows.
- Added the command **sys_DeleteRegKey64**, which deletes the given registry key from the given registry view.
- Added the command **sys_SendRawPrinterData**, which sends the given data directly to the print spooler of the specified printer. Note: On operating systems newer than Windows XP, this command should be used instead of **sys_PrintDirect2Driver**.
- Changed the command **sys_GetPrintJob** so that it resizes the passed array to 10 instead of 1 if it was unable to retrieve the selected printer. In this case, **sys_GetPrintJob** will attempt to fill the array with the default printer's information.
- Added the option to sort the document list returned by **sys_GetDocumentList** by passing 2 for the sort parameter.

New in Version 6.5.1

Bug Fixes

- Fixed a bug that sometimes prevented **gui_GetOpenFileName** from releasing the lock on the directory that was accessed.

New in Version 6.5

Bug Fixes

- Fixed **sys_GetOSVersion** to return the correct Windows version number with Windows 8.1 and Windows Server 2012 R2.

New in Version 6.4.1

Bug Fixes

- Fixed a bug that caused **sys_GetRegText** to crash in 64 bit applications.

New in Version 6.4

Bug Fixes

- Fixed the order in which files are returned from **sys_GetDocumentList** to return the files by oldest creation date first.

Changes

- Added the ability to pass in a starting index to **sys_GetDocumentList** to specify where the list of files should begin.

New in Version 6.3

Bug Fixes

- Fixed a bug when using **gui_RespectToolBar** and maximizing windows.
- Updated **sys_IsAppLoaded** to return the correct value when more than 1024 processes are running.

Changes

- Updated **TWAIN_AcquireImage** to be compatible with all language versions of 4D, not just English. Also, added the ability for users to pass in an optional BLOB parameter that can be used instead of the required **xTWAINBLOB**.

New in Version 6.2.1

Bug Fixes

- Fixed an error where the constant **WIN32API_VERSION** was returning the wrong version number.

- Fixed an error in the command **gui_GetWindowFrom4DWin** that was causing the incorrect window handle to be returned when a process is hidden in Remote Mode. In order to use this command on the server, the user will have to pass in a value of 1 as the second parameter and the process must not be hidden.

New in Version 6.2.0

Bug Fixes

- Fixed a bug that caused Win32API commands to not be recognized in 64-bit applications.

- Fixed a bug that caused the command **gui_GetWindowFrom4DWin** to crash in 64-bit applications.

Changes

- Updated the command **sys_ShellExecute** to accept filenames that contain Unicode characters.

New in Version 6.1.0

Changes

- The command **gui_LoadBackground** is now compatible with 4D v14.

New in Version 6.0

Changes

- Win32API is now developed in Visual Studio 2013 and is no longer compatible with older versions of Visual Studio.

- Added a new command **sys_GetFileVersionInfo** to retrieve the version information of .dll and .exe files.

New in Version 5.3

Bug fixes

- Updated Win32API to keep the TWAIN capture dialog as the front-most window in Harvest.
- Fixed a bug that would cause a 4D client timeout when a TWAIN device dialog was left open for a prolonged period of time.
- Updated Win32API to use an older DLL when checking for running applications when on a version of Windows prior to Vista. Also updated **sys_IsAppLoaded** so that a 64-bit application can enumerate both 64-bit and 32-bit applications.
- Fixed a bug that caused one of the window handles set at start up to get set incorrectly. This bug affected only **gui_SubClassInit**.
- Fixed a bug that caused Harvest Client to crash when clicking the Acquire button on the Links window.
- Fixed a bug that caused the title bar icon of other open programs to change to the Harvest fruit basket.
- Fixed a bug in **sys_GetPrintJob** that caused Harvest to crash. Also fixed similar issues affecting printing and the acquisition of TWAIN images.

Changes

- The command **sys_GetOSVersion** will now recognize Windows 8 and Windows Server 2012.
- Added an optional 6th parameter to **gui_SetWndRect**. Any value greater than 0 passed in this parameter will prevent the window from hiding the task bar.
- Added a new command **sys_CompareBLOBs** to do a byte comparison between the two BLOBs passed in.

New in Version 5.0

Changes

- Now compatible with the 64-bit version of 4D Server.
- The plugin has been rewritten to use the latest plugin API from 4th Dimension.

New in Version 4.1.0

Bug fixes

- Fixed a misspelling in an error code for **sys_ShellExecute**.

Changes

- Added 4 new commands **(sys_SetRegText, sys_SetRegLongint, sys_SetRegArray,** and **sys_SetRegBLOB**) to set Windows registry values.
- Added a new command **sys_IsAppRunningAsService** to determine if 4D is currently running as a Windows service.

New in Version 4.0.0

Bug fixes

- Updated **sys_DirectoryExists** to correctly return false if the specified drive does not exists.
- Fixed several bugs when using **gui_RespectToolbar** and maximizing or minimizing the window.
- Fixed a bug in **gui_SetTrayIcon** that would prevent the icon from being removed under Windows 7.

Changes

- The command **sys_GetOSVersion** will now recognize Windows 7, Windows Server 2008 and Windows Server 2008 Release 2.
- When using **gui_RespectToolbar**, the array <>TB_NOTIFICATION can no longer be used with 4Dv11. Instead a generic call to each process that has an active toolbar will be made.
- Updated multiple commands to work correctly under Windows 7.
- Updated multiple commands to work correctly using 4D v11.
- Added a new command **sys_GetTimeZoneList** to retrieve a list of time zones defined in Windows.
- Add basic support for TWAIN imaging devices. The three commands to utilize this are **TWAIN_GetSources**, **TWAIN_SetSource**, and **TWAIN_AcquireImage**.
- Added a new command **sys_IsAppFrontmost** to determine if 4D is the frontmost application.

- Added a new command **gui_MessageBox** to allow access to the default Windows message boxes.
- Added new commands **gui_HideTaskBar** and **gui_ShowTaskBar** to hide or show the Windows task bar.
- Added new commands **gui_SetMIDOpaque** and **gui_SetMDITransparent** to make the 4D MDI window background opaque or transparent.
- Added new commands **gui_HideTitleBar** and **gui_ShowTitleBar** to hide and show the 4D window title bar.
- Added new commands **gui_MaximizeMDI, gui_MinimizeMDI** and **gui_RestoreMDI** to programmatically maximize, minimize, and restore the 4D MDI window.
- Added new commands **sys_DisableTaskManager** and **sys_EnableTaskManager** to enable or disable the ability of the user to open the Windows task manager.

System Methods

sys_CompareBLOBs

sys_CompareBLOBs(blob1;blob2)→equal

Parameter	Type	Description
blob1	BLOB	[In] BLOB to compare
blob2	BLOB	[In] BLOB to compare
Equal	Longint	[Out] Comparison result

Description

The **sys_CompareBLOBs** command does a byte comparison between the two BLOBs passed in.

Parameters

blob1 – BLOB. This parameter will be compared to the second.

blob2 – BLOB. This parameter will be compared to the first.

Return values

A 0 is returned if both BLOBs are equal and a non-zero value if they are different.

Example

```
C_BLOB($xBLOB1;$xBLOB1)

If (sys_CompareBLOBs($xBLOB1;$xBLOB2)=0)
                // Equal!
End If
```

sys_DecryptAES

sys_DecryptAES(encryptedMessage;password;IV)→decryptedText

Parameter	Type	Description
encryptedMessage	text	[In] Base64 encoded encrypted message to decrypt.
password	text	[In] Password used to encrypt message.
IV	text	[In] Initialization vector to encrypt the message with.
decryptedText	text	[Out] The decrypted text.

Description

The **sys_DecryptAES** function will decrypt the given AES256 encrypted string by using the SHA256 hash of the password as the key and passed IV as the initialization vector. Currently this command only supports the CBC cipher mode.

Parameters

encryptedMessage – text. This is the base64 message to decrypt. The maximum length is 256 characters.

password – text. Win32API will compute the SHA256 hash of this password and use the hash as the key. The maximum length is 32 characters.

IV – text. This text is used as the initialization vector. You should always pass 16 characters here. Note that without the IV used to encrypt the message, you will be unable to decrypt it.

decryptedText – text. The ASCII encoded decrypted text. The maximum length is 256 characters.

Example

```
C_TEXT($tEncMes;$tDecMes;$tIV)

$tIV:="0123456789ABCDEF"

$tEncMes:=sys_EncryptAES("This is an example of encryption using
Win32API";"ZXYJ1234";$tIV)
$tDecMes:=sys_DecryptAES($tEncMes;"ZXYJ1234";$tIV)
```

sys_DeleteRegKey

sys_DeleteRegKey(rootkey;subKey)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being deleted.
subKey	text	[In] The subkey of the registry key being deleted.
errorCode	longint	[Out] Error code.

Description

The **sys_DeleteRegKey** command deletes the specified registry key. Note: When used on 64-bit Windows, this command will only delete the registry key from the 32-bit registry view.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being deleted.

Error Codes

If the function fails the error code return value is non-zero. If the function succeeds, the error code return value is zero.

Example

```
C_LONGINT($lErr)
$lErr:=sys_DeleteRegKey(GR_HKEY_CURRENT_USER;"Software\\Applicati
on")
```

sys_DeleteRegValue

sys_DeleteRegValue(rootKey;subKey;name)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key of the value being deleted.
subKey	text	[In] The subkey of the registry key of the value being deleted.
name	text	[In] The name of the value being deleted.
errorCode	longint	[Out] Error code.

Description

The **sys_DeleteRegValue** command deletes the specified registry value.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry value being deleted.

name – text. This parameter is the name of the value being deleted.

Error Codes

If the function fails the error code return value is non-zero. If the function succeeds, the error code return value is zero.

sys_DirectoryExists

sys_DirectoryExists(directorypath)→exists

Parameter	Type	Description
directorypath	text	[In] The path of the directory.
exists	longint	[Out] File exists or not.

Description

The **sys_DirectoryExists** function checks for the existence of a directory.

Parameters

filename – text. This parameter is the fully qualified directory path.

Return values

If the directory is found, then a 1 is returned; otherwise a 0 is returned.

Example

```
C_BOOLEAN($bDirectoryExists)
C_TEXT($1;$tDirectoryPath)

$tDirectoryPath:=$1
$bDirectoryExists:=(sysDirectoryExists($tDirectoryPath)=1)
if($bDirectoryExists)
  // Do some action with the file
end if
```

sys_DisableTaskManager

sys_DisableTaskManager→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **sys_DisableTaskManager** function disables the ability of the user to open the Windows task manager. Users will be unable to open the task manager until a call to sys_EnableTaskManager is made.

Error Codes

If the task manager was successfully disabled the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=sys_DisableTaskManager


If ($lErr#0 )
        ` Users can no longer open the Windows task manager.
End if
```

sys_EnableTaskManager

sys_EnableTaskManager→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **sys_EnableTaskManager** function enables the ability of the user to open the Windows task manager.

Error Codes

If the task manager was successfully enabled the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=sys_EnableTaskManager


If ($lErr#0 )
        ` Users can one again open the Windows task manager.
End if
```

sys_EncryptAES

sys_EncryptAES(message;password;IV)→encryptedText

Parameter	Type	Description
message	text	[In] Message to encrypt.
password	text	[In] Password to encrypt message with.
IV	text	[In] Initialization vector to encrypt the message with.
encryptedText	text	[Out] The encrypted text, encoded in Base64.

Description

The **sys_EncryptAES** function will encrypt the given message in AES256 by using the SHA256 hash of the password as the key and passed IV as the initialization vector. Currently, this command only supports the CBC cipher mode.

Parameters

message – text. This is the message to encrypt. The maximum length is 256 characters.

password – text. Win32API will compute the SHA256 hash of this password and use the hash as the key. The maximum length is 32 characters.

IV – text. This text is used as the initialization vector. You should always pass 16 characters here. Note that without the IV used to encrypt the message, you will be unable to decrypt it.

encryptedMessage – text. This is the encrypted message. It will be encoded in Base64.

Example

```
C_TEXT($tEncMes;$tDecMes;$tIV)

$tIV:="0123456789ABCDEF"

$tEncMes:=sys_EncryptAES("This is an example of encryption using
Win32API";"ZXYJ1234";$tIV)
```

sys_EnumPrinters

sys_EnumPrinters(printerList;format)→errorCode

Parameter	Type	Description
printerList	text array	[Out] Array of printers as defined on the workstation.
format	longint	[In] Optional constant to specify string format.
errorCode	longint	[Out] Error code.

Description

The **sys_EnumPrinter** call retrieves an array of all printers available for the workstation. The array element can be obtained in different ways (and therefore, in different formats) if the second parameter is used.

Parameters

printerList – text array. This is the text array variable initialized to zero elements. It is used to receive the list of printers.

format – optional. Constant determining format of printer info.

Constant	Description
EP_USE_REGISTRY (0)	This is the default format. For Win 9x and Me, the win.ini file is used because printer information is stored here. For NT, Win2K and XP, the registry is used. Local printer information is returned as <printer name>,<spooler>,<port>. Network printer information is returned as <printeruncname>,<spooler>,<port>. Example: MyPrinter,HPPCL5MS,LPT1
EP_NAMES_ONLY (1)	Only the printer names are returned in the array elements. This format may not be used to set the default printer using the Win32API function sys_SetDefPrinter. Use this format to present an easier to read list of printers to the user. Then, search for this string inside the array of printers returned by EP_USE_REGISTRY.
EP_USE_OPEN (2)	This is the format string used in previous versions of the Win32API plugin. Its use is not recommended, and it is maintained only for backwards compatibility.

Error Codes

When the constant EP_USE_REGISTRY is used, the error code is the array element number containing the default printer. Otherwise, if the function succeeds, the error code return value is equal to the number of printers in the list. If the function fails, the error code return value is zero.

Example

```
C_LONGINT($lErr;$i)
C_TEXT($tPtrNames)
ARRAY TEXT(atPrinters;0)

$lErr:=sys_EnumPrinters (atPrinters)
If ($lErr#0)
  If (Size of array(atPrinters)>0)
    For ($i;1;Size of array(atPrinters))
      $tPtrNames:=$tPtrNames+atPrinters{$i}+Char(13)
    End for
    ALERT("Printers found:  "+Char(13)+$tPtrNames)
  Else
    ALERT("No printers registered.")
  End if
End if
```

sys_EnumProcesses

sys_EnumProcesses(processNames;processIDs) →errorCode

Parameter	Type	Description
processNames	array text	[Out] The names of the processes running on the machine.
processIDs	array longint	[Out] The IDs of the processes running on the machine.
errorCode	longint	[Out] Error code.

Description

The **sys_EnumProcesses** function enumerates the processes running on the machine into parallel arrays of names and IDs.

Parameters

processNames – array text. This parameter will receive the names of each process running on the machine.

processIDs – array longint. This parameter will receive the IDs of each process running on the machine. It will be parallel to the *processNames* array.

Error Codes

If the function succeeds, the error code return value is zero. If the function fails, the error code return value will be set to the Windows error code that corresponds to the generated error.

Example

```
C_LONGINT($lErr;$lPos)
C_TEXT($1;$tProcName)
ARRAY TEXT(atProcNames;0)
ARRAY LONGINT(atProcIDs;0)

$lErr:=sys_EnumProcesses(atProcNames;atProcIDs)
if($lErr=0)
  $lPos:=Find in array(atProcNames;$tProcName)
  if($lPos>0)
    $lErr:=sys_KillProcessById(atProcIDs{$lPos};0)
  end if
end if
```

sys_FileCheck

sys_FileCheck(cmpltPathFileName)→errorCode

CAUTION: This function is included for completeness in the documentation. It has a very specific purpose (see description below). This function should only be used as described below. Using this function on a file not currently open with Exclusive access will delete the file.

Parameter	Type	Description
cmpltPathFilename	text	[In] Complete path and file name.
errorCode	text	[Out] Error code.

Description

The **sys_FileCheck** function tries to create the file and then returns an error code status. This function is used to obtain a 'signal' that a second (different) program has quit. The second application creates a special file that is automatically deleted when the application quits or is terminated. The **sys_FileCheck** function attempts to create the file. If the file is created, then it can safely be assumed that the second application has terminated. The file created by the plugin call is then deleted.

Parameters

cmpltPathFilename – text. This is a text variable containing the name of the file that **sys_FileCheck** will attempt to create.

Error Codes

The function returns the error code obtained by trying to create the file.

ERROR_FILE_NOT_FOUND (2)

ERROR_PATH_NOT_FOUND (3)

ERROR_ACCESS_DENIED (5)

ERROR_SHARING_ VIOLATION (32)

ERROR_FILE_EXISTS (80)

Example

```
C_LONGINT($lErr)

$lErr:=sys_FileCheck ("c:\someFolder\filename")
If ($lErr=ERROR_SHARING_VIOLATION)
  `second app has quit
  `do whatever
End if
```

sys_FileExists

sys_FileExists(filepath)→exists

Parameter	Type	Description
filepath	text	[In] The path to the file.
exists	longint	[Out] File exists or not.

Description

The **sys_FileExists** checks for the existence of a file. It is a replacement for the Test Pathname command, as the 4D method has difficulty locating files that begin with ".".

Parameters

filename – text. This parameter is the fully qualified path to a file.

Return values

If the file is found, then a 1 is returned; otherwise a 0 is returned.

Example

```
C_BOOLEAN($bFileExists)
C_TEXT($1;$tFilePath)

$tFilePath:=$1

$bFileExists:=(sysFileExists($tFilePath)=1)
if($bFileExists)
  // Do some action with the file
end if
```

sys_GetCommandLine

sys_GetCommandLine(parameters;action)→errorCode

Parameter	Type	Description
parameters	text array	[Out] Zero-element array to hold parsed command line parameters.
action	longint	[In] Constant indicating that the command line should be parsed for drag and drop.
errorCode	longint	[Out] Error code.

Description

The **sys_GetCommandLine** function parses the command line used to start 4D. Parameters may be passed to 4D at startup using this function if 4D is started from the Run menu, a shortcut, or by 'dropping' a file on to the 4D shortcut.

Parameters

parameters – text array. This is the text array variable initialized to zero elements. It is used to receive the parsed command line parameters.

action – optional. This is a constant that indicates that the command line should be parsed for drag and drop.

Constant	Description
CL_DRAGDROP (1)	Optional. Use this when it is anticipated that 4D will be started by a drag and drop action (see remarks).

Error Codes

The function returns zero if the function fails to read the command line. On success, it returns the number of array elements created.

Remarks

The array elements returned will be as follows:

The element zero will contain the entire command line – including the path/file name for 4D (client or single user). There may be occasions when you want to parse it your way.

If the optional second parameter is not used, each element (after zero) will be a parsed segment of the command line after the path/file name of the 4D component (client or single user). It is expected that each command-line argument is space separated.

Using CL_DRAGDROP causes the command line to be parsed into two array elements. The first is the 4D executable. The second is whatever is left on the command line. If drag & drop is used, it will be the full file name of the file dragged onto the 4D executable. If it is not a drag & drop operation, you will get the rest of

the command line as it was passed to 4D. For example: c:\…\4DClient.exe Joe 3 Jane gives you *Joe 3 Jane* as the second array element.

The second parameter is needed because after the executable, nothing is double quoted -- so when a file is dragged to 4D and there are spaces in the path or file name, they would normally be parsed as parameters.

Note that some users have reported problems with this command. We've found that the memory buffer that contains the command line parameters is corrupted by either 4D or Windows when 4D starts. We've added some special code to work around this corruption, but if any of the paths or command line arguments contain double-quotes you may get unexpected results with this command. Unfortunately, there's nothing we can do to resolve this problem.

Example 1

```
C_LONGINT($lErr;$i)
ARRAY TEXT($atArguments;0)

$lErr:=sys_GetCommandLine ($atArguments)
If ($lErr>0)
  For ($i;1;$lErr)
    `do something
  End for
End if
```

Example 2

```
C_LONGINT($lErr;$i)
ARRAY TEXT($atArguments;0)
C_TEXT($tFileName)

$lErr:=sys_GetCommandLine ($atArguments;CL_DRAGDROP )
If ($lErr=2)
  ` do something with file that was dragged to 4D.exe
  $tFileName:=$atArguments{2}
End if
```

sys_GetDefPrinter

sys_GetDefPrinter(printerName)→errorCode

Parameter	Type	Description
printerName	text	[Out] Printer device name.
errorCode	longint	[Out] Error code.

Description

The **sys_GetDefPrinter** call retrieves the name of the current Windows default printer.

Parameters

printerName – text. After the call is performed, this contains the name of the current printer in the following format:

<printer name>,<driver name>,<port>

For example:

MyPrinter,HPPCL5MS,LPT1

Print server example:

\\orchardsoft\MyPrinter X123,MyPrinter X123,192.168.0.1

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code is zero.

Example

```
C_LONGINT($lErr)
C_TEXT($tCurrentPrinter)
$lErr:=sys_GetDefPrinter ($tCurrentPrinter)
```

sys_GetDocumentList

sys_GetDocumentList(path;filePattern;fileNames;maxFiles;fileSort;startIndex)→errorCode

Parameter	Type	Description
path	text	[In] The search directory.
filePattern	text	[In] The search file pattern.
fileNames	array text	[Out] Array of file names found.
maxFiles	longint	[Out] The maximum number of files to return.
fileSort	longint	[In] The sort type to use. If no sort is specified, the directory sort is used.
startIndex	longint	[In] The directory index to start from. If no start index is specified, startIndex will start at the first file in the directory.
errorCode	longint	[Out] Error code.

Description

The **sys_GetDocumentList** command searches the specified directory for the specified file pattern, and returns the results in a text array. Unlike 4D's **DOCUMENT LIST** command, the number of files returned can be limited.

Parameters

path – text. This parameter is the path in which to search for files.

filePattern – text. This parameter is the file pattern used to filter the results. Any valid DOS file pattern will work with the command. For example, "*.txt" returns only .txt files, and "*.*" or "" returns all files, up to the limit.

fileName – array text. This parameter is an array of file names found by the command.

maxFiles – longint. This parameter is the maximum number of files to return. Set the parameter to -1 to return all found files.

fileSort – longint. Set to "0" use the sort that the directory path is using, set to "1" to sort by oldest creation date first, set to "2" to sort by ascending alphabetical order.

startIndex – longint. This parameter is optional and *fileSort* must be set to use this parameter. It is the starting index for the document list to start from. If the parameter is not set, the list will start from the first item in the directory.

Error Codes

If the function succeeds, the error code return value is zero. If the function fails, the error code return value will be set to the Windows error code that corresponds to the generated error.

Example

```
C_LONGINT($lErr)
ARRAY TEXT($atReturnedFiles;0)

` Get all the files the start with "fresc"
$lErr:=sys_GetDocumentList
("c:/acesulfame/";"fresc*.*";$atReturnedFiles;-1)
```

sys_GetEnv

sys_GetEnv(name;value)→errorCode

Parameter	Type	Description
name	text	[In] The name of the environment variable.
value	text	[Out] The value of the environment variable.
errorCode	longint	[Out] Error code.

Description

The **sys_GetEnv** command will return the value of the supplied environment variable in the *value* parameter.

Parameters

name – text. This parameter is the name of the environment variable to look up.

value – text. This parameter will contain the value of the environment variable.

Error Codes

The function will return one if the environment variable is found. The function will return zero if the environment variable is not found.

Remarks

The function only affects the environment variables of the calling process, not the global Windows variables.

Example

```
C_LONGINT($lErr)
C_TEXT($tComputerName)
C_TEXT($tSystemRoot)

` Get the name of the computer.
$lErr:=sys_GetEnv("COMPUTERNAME";$tComputerName)

` Get the root location of the Windows system directory.
$lErr:=sys_GetEnv("SYSTEMROOT";$tSystemRoot)
```

sys_GetFileVersionInfo

sys_GetFileVersionInfo(filePath;majorNumber;minorNumber;buildNumber;revisionNumber)→errorCode

Parameter	Type	Description
filePath	text	[In] The path of the .dll or .exe file.
majorNumber	longint	[Out] The major number of the file.
minorNumber	longint	[Out] The minor number of the file.
buildNumber	longint	[Out] The build number of the file.
revisionNumber	longint	[Out] The revision number of the file.
errorCode	longint	[Out] Error code.

Description

The **sys_GetFileVersionInfo** call returns the major, minor, build, and revision numbers of the .dll or .exe file passed in.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Example

```
C_LONGINT($lErr, $lMajor, $lMinor, $lBuild, $lRevision)

$lErr:=sys_GetFileVersionInfo("C:\\4D_V13\\4D.exe";$lMajor;$lMinor;$lBuild;$lRevision)

If($lErr#0)
  ALERT("File version:
  "+String($lMajor)+"."+String($lMinor)+"."+String($lBuild)+"."+String($lRevision))
End if
```

sys_GetGUID

sys_GetGUID(guid;qualifier)→errorCode

Parameter	Type	Description
guid	text	[Out] Global unique identifier.
qualifier	text	[Out] Qualifies uniqueness.
errorCode	longint	[Out] Error code.

Description

The **sys_GetGUID** call creates a global unique identifier. The qualifier indicates if the system imposed a restriction such as uniqueness limited to a local workstation.

Parameters

guid – text. After the call is performed, this contains a unique identifier formatted as: one group of 8 hex digits followed by three groups each containing 4 hex digits, followed by one group of 12 hex digits.

Example:

```
6B29FC40-CA47-1067-B31D-00DD010662DA
```

qualifier – text. Computers without a network card may or may not allow the generation of a GUID. Qualifiers returned by this command other than 'OK' are: 'Local only' – meaning that the GUID may not be unique if used on a different computer, or 'Cannot get Ethernet hardware address' – meaning that there is no network card, or the network card cannot be addressed by the routine.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code is zero.

Example

```
C_LONGINT($lErr)
C_TEXT($tGuid;$tQualifier)
$lErr:=sys_GetGUID ($tGuid;$tQualifier)
ALERT("GUID: "+$tGuid+Char(13)+$tQualifier)
```

sys_GetNetworkInfo

sys_GetNetworkInfo(infoString)→errorCode

COMPATIBILITY: Available on Windows 98, Windows 2000, and Windows XP. Not available on NT 3.51, NT 4.0, or Windows 95.

Parameter	Type	Description
infoString	text variable	[Out] Comma delimited network info.
errorCode	longint	[Out] Error code.

Description

The **sys_GetNetworkInfo** call returns information similar to what one would get using IPConfig. Information is in the following order.

Host name

Domain name

IP of domain server

Type (broadcast, peer-to-peer, mixed, or hybrid)

Used for routing

Used as DNS server

Acting as ARP proxy

List of DNS servers used (may be none to many)

Parameters

infoString – text variable. After the call is performed, this parameter contains information as listed under description.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code is zero.

Example

```
C_TEXT($tNetString)
$lErr:=sys_GetNetworkInfo ($tNetString)
ALERT($tNetString)
```

sys_GetOneRegionSetting

sys_GetOneRegionSetting(regionSetting;specificInfo)→errorCode

Parameter	Type	Description
regionSetting	text	[Out] Return value.
specificInfo	longint	[In] Region information to retrieve.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegionSettings** and **sys_GetOneRegionSetting** calls retrieve system formatting information as defined for the current user.

Parameters

regionSetting – This is the text buffer used to receive the string information.

specificInfo – This is a constant to determine the specific information to be retrieved.

Constant	Description
RS_SHORTDATEFORMAT (1)	Short date format consists of a combination of month, day, & year format pictures. Example: M/d/yyyy
RS_LONGDATEFORMAT (2)	Long date format consists of a combination of month, day, year, and era format pictures. Example: dddd, MMMM dd, yyyy (Monday, June 23, 2005)
RS_DATESEPARATOR (3)	Character(s) used as the date separator.
RS_TIMEFORMAT (4)	Time format consists of a combination of hour, minute, & second format pictures. Example: hh:mm:ss
RS_TIMESEPARATOR(5)	Character(s) used as the time separator.
RS_AMSYMBOL (6)	String for the AM designator.
RS_PMSYMBOL (7)	String for the PM designator.
RS_MEASURESYSTEM (8)	Metric or U.S.
RS_DECIMALSYMBOL (9)	Character used as decimal separator.

RS_NUMBERLEADINGZEROS (10)

Zero (Example: .5) or one (Example: 0.5).

RS_DIGITSAFTERDECIMAL (11)

Number of digits after decimal.

RS_NUMBERGROUPINGSYMBOL (12)

Character(s) used to separate digit groups to the left of the decimal.

RS_NEGATIVESYMBOL (13)

String value for the negative sign.

RS_CURRENCYSYMBOL (14)

String used for the local monetary symbol.

RS_CURRENCYDECIMALSYMBOL (15)

Character used as the decimal separator in currency.

RS_CURRENCYDIGITSAFTERDECIMAL (16)

Number of digits after the monetary decimal.

RS_CURRENCYGROUPINGSYMBOL (17)

Character(s) used to separate digit groups to the left of the decimal in monetary numbers.

RS_LISTSEPARATOR (18) Character(s) used to separate list items.

Error Codes

If the function succeeds, the error code return value is the length of the text. If the function fails, the error code return value is zero.

Example

```
C_TEXT($tText)
C_LONGINT($lErr)
$tText:=""
$lErr:=sys_GetOneRegionSetting ($tText;RS_LongDateFormat )
ALERT("The region setting requested is: "+$tText)
```

sys_GetOSVersion

sys_GetOSVersion(version;moreInfo)→errorCode

Parameter	Type	Description
version	longint	[Out] Number corresponding to constant listed below.
moreInfo	text	[Out] Extra text providing more information about the operating system. For Windows NT/2000, this will be the service pack installed. For Windows 95/98, this will be a letter (e.g., "B" for Windows 95 B).
errorCode	longint	[Out] Error code.

Description

The **sys_GetOSVersion** call returns the Windows Operating System version.

Parameters

version – longint. This variable receives a number that corresponds to the following constants.

Constant	Description
OS_WIN95 (1)	Windows 95
OS_WIN98 (2)	Windows 98
OS_ME (3)	Windows Me
OS_NT351 (351)	Windows NT 3.51
OS_NT4 (400)	Windows NT 4
OS_W2K (500)	Windows 2000
OS_XP (510)	Windows XP
OS_WIN03 (520)	Windows Server 2003
OS_VISTA_LONGHORN (600)	Windows Vista or Longhorn
OS_SERVER2K8 (601)	Windows Server 2008
OS_WIN7 (610)	Windows 7
OS_WIN8 (620)	Windows 8
OS_WIN81 (630)	Windows 8.1
OS_SERVER2K8R2 (611)	Windows Server 2008 Release 2
OS_SERVER2012 (621)	Windows Server 2012
OS_SERVER2012R2 (631)	Windows Server 2012 Release 2

moreInfo – text. Extra information about the operating system.

Error Codes

If the function succeeds, the error code return value is positive. If the function fails, the error code return value is zero.

Example

```
C_TEXT($tServicePack)
C_LONGINT($lErr;$lVersion)
$lErr:=sys_GetOSVersion ($lVersion;$tServicePack)
If ($lVersion=OS_XP )
  ALERT("Windows XP")
End if
```

sys_GetPrintJob

sys_GetPrintJob(printer;4DCommand)→errorCode

Parameter	Type	Description
printer	text array	[Out] Empty text array.
4DCommand	text	[In] Executable 4D command that calls the two print dialogs. If no command is supplied, "Print Settings" is the default.
errorCode	longint	[Out] Error code.

Description

The **sys_GetPrintJob** command is a replacement (wrapper) method for 4D's PRINT SETTINGS command. After you call **sys_GetPrintJob**, you will be able to retrieve information selected by the user in the Windows Page Setup dialog boxes, such as the name of the printer that the user selected, the number of copies, and more.

Seven parameters are returned in a text array. You can reference these array elements by the constants listed in the Remarks for this command.

The PRINT SETTINGS command is the default command that is wrapped by this plugin call. The plugin wraps PRINT SETTINGS by name, and not by the internal 4D command ID number. If you are using a localized version of 4D, you should specify the localized command name that corresponds to PRINT SETTINGS in the English version of 4D. If you do not pass the second parameter, PRINT SETTINGS will be called.

Parameters

printer – text array. A seven-element array is returned with information from the two print dialog boxes displayed by 4D.

4DCommand – text. This variable is optional. If not used, the default command is "Print Settings."

Error Codes

If the function succeeds, the error code return value is non-zero. Zero is returned if the call fails.

Remarks

Constants can be used to retrieve information from the array. The index constants are:

Constant	Description
PS_PRINTER (1)	Printer Name
PS_SIZE (2)	Paper Size
PS_SOURCE (3)	Tray Info
PS_COPIES (4)	Number of Copies as String
PS_PORTRAITORLANDSCAPE (5)	"Portrait" or "Landscape"
PS_PRINTEDTOFILE (6)	"Printed to File" or " "
PS_PRINTPREVIEW (7)	"Print Preview" or " "

Example

```
C_LONGINT($lErr)
C_TEXT($tPrinterName)
ARRAY TEXT($atPrintInfo;0)
  ` Load the application's default printer options
PAGE SETUP([Dialogs];"defaultPageSetup")
  ` Present the PRINT SETTINGS dialog
$lErr:=sys_GetPrintJob ($atPrintInfo)
If (OK=1)  ` The user clicked the OK button
  If ($lErr=0)  ` There was an error in the plugin call
    $tPrinterName:="Unknown"
  Else
    $tPrinterName:=$atPrintInfo{PS_Printer }
  End if
  ALERT("The selected printer was "+$tPrinterName)
Else   ` The user clicked Cancel
  ALERT("The PRINT SETTINGS dialog was cancelled.")
End if
```

sys_GetRegArray

sys_GetRegArray(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
value	text array	[Out] The value of the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegArray** command queries for the specified registry entry of type array. The value of the entry will be returned in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – text array. This parameter returns the value of the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type array to work with this command.

sys_GetRegBlob

sys_GetRegBlob(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
value	blob	[Out] The value of the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegBlob** command queries for the specified registry entry of type binary. The value of the entry will be returned in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – blob. This parameter returns the value of the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type binary to work with this command.

sys_GetRegEnum

sys_GetRegEnum(rootKey;subKey;keys;names)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
keys	array text	[Out] Array of subkeys in the specified registry.
names	array text	[Out] Array of value names in the specified registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegEnum** command queries for the specified registry key and returns all of the subkeys and value names contained within the key.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

keys – array text. When the function returns, this parameter will contain all the subkeys contained in the specified key.

namess – array text. When the function returns, this parameter will contain all the value names contained in the specified key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Example

```
C_LONGINT($lErr)
ARRAY TEXT($atSubkeys;0)
ARRAY TEXT($atNames;0)

` Get all the subkeys and values contained in
GR_HKEY_CLASSES_ROOT.

$lErr:=sys_GetRegEnum (GR_HKEY_CLASSES_ROOT ;"";$
atSubkeys;$atNames)
```

sys_GetRegionSettings

sys_GetRegionSettings(regionSetting;settingDescription)→errorCode

Parameter	Type	Description
regionSetting	text array	[Out] Regional information.
settingDescription	text array	[Out] Description of information.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegionSettings** call retrieves system formatting information as defined for the current user.

Parameters

regionSetting – This is a text array used to receive the string information. Use the constants below to iterate through the array.

settingDescription – A description of the region setting. Listed in {} in action column below.

Constant	Description
RS_SHORTDATEFORMAT (1)	Short date format consists of a combination of month, day, & year format pictures. Example: M/d/yyyy {Short Date}
RS_LONGDATEFORMAT (2)	Long date format consists of a combination of month, day, year, and era format pictures. Example: dddd, MMMM dd, yyyy (Monday, June, 23, 2005). {Long Date}
RS_DATESEPARATOR (3)	Character(s) used as the date separator. {Date Separator}
RS_TIMEFORMAT (4)	Time format consists of a combination of hour, minute, & second format pictures. Example: hh:mm:ss {Time Format}
RS_TIMESEPARATOR (5)	Character(s) used as the time separator. {Time Separator}
RS_AMSYMBOL (6)	String for the AM designator. {AM Symbol}
RS_PMSYMBOL (7)	String for the PM designator. {PM Symbol}
RS_MEASURESYSTEM (8)	Metric or U.S. {Measurement System}
RS_DECIMALSYMBOL (9)	Character used as decimal separator. {Decimal Symbol}

RS_NUMBERLEADINGZEROS (10)

Zero (Example: .5) or one (Example: 0.5). {Leading Zeros}

RS_DIGITSAFTERDECIMAL (11)

Number of digits after decimal. {Digits after Decimal}

RS_NUMBERGROUPINGSYMBOL (12)

Character(s) used to separate digit groups to the left of the decimal. {Number Grouping Symbol}

RS_NEGATIVESYMBOL (13)

String value for the negative sign. {Negative Symbol}

RS_CURRENCYSYMBOL (14)

String used for the local monetary symbol. {Currency Symbol}

RS_CURRENCYDECIMALSYMBOL (15)

Character used as the decimal separator in currency. {Currency Decimal Symbol}

RS_CURRENCYDIGITSAFTERDECIMAL (16)

Number of digits after the monetary decimal. {Currency Digits after Decimal}

RS_CURRENCYGROUPINGSYMBOL (17)

Character(s) used to separate digit groups to the left of the decimal in monetary numbers. {Currency Grouping Symbol}

RS_LISTSEPARATOR (18) Character(s) used to separate list items. {List Separator}

Error Codes

If the function succeeds, the error code return value is the number of elements of the array. If the function fails, the error code return value is zero.

Example

```
C_TEXT($tText)
C_LONGINT($lErr)
ARRAY TEXT(atRegionArray;0)
ARRAY TEXT(atDescrArray;0)
$lErr:=sys_GetRegionSettings (atRegionArray;atDescrArray)
If (Size of array(atRegionArray)>0)
  ALERT(atDescrArray{RS_LongDateFormat }+" is
        "+atRegionArray{RS_LongDateFormat })
End if
```

sys_GetRegLongint

sys_GetRegLongint(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
value	longint	[Out] The value of the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegLongint** command queries for the specified registry entry of type longint. The value of the entry will be returned in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – longint. This parameter returns the value of the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type longint to work with this command.

sys_GetRegText

sys_GetRegText(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
value	text	[Out] The value of the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegText** command queries for the specified registry entry of type text. The value of the entry will be returned in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key.

Constant	Description
GR_HKEY_CLASSES_ROOT (1)	Typically contains file extension associations, and is primarily intended for compatibility with the registry in 16-bit Windows.
GR_HKEY_CURRENT_USER (2)	Typically contains information about the current user hardware profile.
GR_HKEY_DYN_DATA (3)	Typically contains performance data for Windows Me/98/95 machines.
GR_HKEY_LOCAL_MACHINE (4)	Typically contains hardware and software settings for a machine.
GR_HKEY_USERS (5)	Typically contains the default user configuration for new users.
GR_HKEY_CURRENT_CONFIG (6)	Typically contains information about the current hardware profile of the local computer system.

GR_HKEY_PERFORMANCE_DATA (7)

Typically contains performance data for non Windows Me/98/95 machines.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – text. This parameter returns the value of the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type text to work with this command. Registry values that contain references to environment variables, such as "%SYSTEMPATH%," will be expanded, and the references will be replaced by the environment variables' defined values.

Example

```
C_LONGINT($lErr)
C_TEXT($tText)

` Query for the specified key.
$lErr:=sys_GetRegText (GR_HKEY_CURRENT_USER ;"Control
Panel\\Accessibility\\HighContrast";"High Contrast
Scheme";$tText)
```

sys_GetRegType

sys_GetRegType(rootKey;subKey;name)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRegType** command queries for the specified registry key and returns its data type.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

Error Codes

If the function fails, the error code return value is zero. If the function succeeds, the error code return value is greater the zero:

Constant	Description
GR_TYPE_BINARY (1)	Binary data in any form.
GR_TYPE_LONGINT (2)	A 32-bit number.
GR_TYPE_TEXT (3)	A text value.
GR_TYPE_ARRAYTEXT (4)	A text array.

sys_GetRoutes

sys_GetRoutes(atRouteInfo) →errorCode

Parameter	Type	Description
atRouteInfo	text array	[Out] Comma delimited route table info.
errorCode	longint	[Out] Error code.

Description

The **sys_GetRoutes** call retrieves a comma-delimited string for each row in the route table. The order of the info in each string is:

Destination IP

Subnet of destination IP

IP of next hop

Route type (NA, Invalid, Local, or Remote)

Number of seconds since route was last used or verified

Parameters

atRouteInfo – text array. After the call is performed, there is one element for each row in the route table.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code is zero.

Example

```
ARRAY TEXT($atRouteInfo;0)
C_LONGINT($i;$x)
C_TEXT($tMsg)
$lErr:=sys_GetRoutes ($atRouteInfo)
$x:=Size of array($atRouteInfo)
$tMsg:="Number of routes: "+String($x)+Char(13)
For ($i;1;$x)
  $tMsg:=$tMsg+$atRouteInfo{$i}+Char(13)
End for
ALERT($tMsg)
```

sys_GetTimeZone

sys_GetTimeZone(standardTime;daylightTime;autoAdjForDaylight)→errorCode

Parameter	Type	Description
standardTime	text	[Out] Standard time zone.
daylightTime	text	[Out] Daylight time zone.
autoAdjForDaylight	longint	[Out] Flag to adjust time for seasonal time changes.
errorCode	longint	[Out] Error code.

Description

The **sys_GetTimeZone** call retrieves the standard time and daylight time descriptions.

Parameters

standardTime – text. This is the buffer used to receive the string containing the description of the standard time zone.

daylightTime – text. This is the buffer used to receive the string containing the description of the daylight time zone.

autoAdjForDaylight – longint. This is set to one if the "Automatically adjust clock for daylight savings changes" is checked, otherwise zero.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Example

```
C_LONGINT($lErr;$lAuto)
C_TEXT($tStandard;$tDaylight)
$lErr:=sys_GetTimeZone ($tStandard;$tDaylight;$lAuto)
$tMsg:="Time zone info:  Standard: "+$tStandard
$tMsg:=$tMsg+"  Daylight: "+$tDaylight+Char(13)
$tMsg:=$tMsg+"AutoDaylight: "+String($lAuto)
ALERT($tMsg)
```

sys_GetTimeZoneList

sys_GetTimeZoneList (timeZones)→errorCode

Parameter	Type	Description
timeZones	array text	[Out] Array of time zones returned by Windows.
errorCode	longint	[Out] Error code.

Description

The **GetTimeZoneList** call returns a text array of all time zones defined on the workstation. Each array element is in the following format:

TimeZoneName;CurrentTime;CurrentDate

Parameters

timeZones – text array. This is the text array variable initialized to zero elements. It is used to receive the list of time zones.

Error Codes

If the function succeeds, the error code return value is equal to the number of time zones returned. If the function fails, the error code return value is less than zero.

Example

```
C_LONGINT($lErr;$i)
C_TEXT($tTimeZones)
ARRAY TEXT($atTimeZones;0)

$lErr:=sys_GetTimeZoneList ($atTimeZones)

If ($lErr>0)

      For ($i;1;$lErr)
            $tTimeZones:=$tTimeZones+$atTimeZones{$i}+Char(13)
      End for
      ALERT("Time Zones found:"+Char(13)+$tTimeZones)
End if
```

sys_GetUserName

sys_GetUserName(userName)→errorCode

Parameter	Type	Description
userName	string	[Out] Current user name.
errorCode	longint	[Out] Error code.

Description

The **sys_GetUserName** call retrieves the user name of the current thread. This is the login name of the user currently logged onto the system.

Parameters

userName – string. This is the buffer used to receive the string containing the user's login name. If the real login name is longer than 250 characters, then only the first 250 characters will be returned.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Important Note: The error code was changed in version 3.0 to be consistent with the other methods. Previous versions returned zero if the function succeeded and non-zero if the function failed.

Example

```
C_LONGINT($lErr)
C_STRING(254;$s254CurrentUserName)
$lErr:=sys_GetUserName ($s254CurrentUserName)
```

sys_GetUTCOffset

sys_GetUTCOffset(minutes)→errorCode

Parameter	Type	Description
minutes	longint	[Out] Minutes from GMT.
errorCode	longint	[Out] Error code.

Description

The **sys_GetUTCOffset** call retrieves the number of minutes from Universal Coordinated Time (GMT). This is based upon the time zone setting of the computer.

Parameters

minutes – longint. Will contain the difference in minutes between local time and GMT.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

If the change based upon daylight savings time is to be compensated for, use **sys_GetTimeZone** to see if the autoAdjForDaylight flag is zero.

Example

```
C_LONGINT($lErr;$lMinutes)
$lErr:=sys_GetUTCOffset ($lMinutes)
ALERT("Number of minutes from GMT is: "+String($lMinutes))
```

sys_GetWindowMetrics

sys_GetWindowMetrics(metricRequest)→metricValue

Parameter	Type	Description
metricRequest	longint	[In] Constant indicating what metric is to be returned.
metricValue	longint	[Out] Return value in pixels. If function fails, zero is returned.

Description

The **sys_GetWindowMetrics** command returns the pixel heights of various Windows user interface components, such as the menu bar height, window border width, and title bar height.

Parameters

metricRequest – longint. One of the constants listed below.

Constant	Description
WM_BORDER_WIDTH (5)	Window border width
WM_BORDER_HEIGHT (6)	Window border height
WM_CAPTION_HEIGHT (4)	Window caption (title bar) height
WM_MENU_HEIGHT (15)	Menu height

Error Codes

If the function succeeds, the error code return value is in pixels. If the function fails, the error code return value is zero.

Example

```
C_LONGINT($lMetricValue;$lMetricRequest)
$lMetricRequest:=WM_CAPTION_HEIGHT   `window title bar
$lMetricValue:=sys_GetWindowMetrics ($lMetricRequest)
inutes from GMT is: "+String($lMinutes))
```

sys_IsAppFrontmost

sys_IsAppFrontmost→returnCode

Parameter	Type	Description
returnrCode	longint	[Out] 1 if the application is the frontmost Windows application, 0 if it is not.

Description

The **sys_IsAppFrontmost** checks to see if the calling application is the foreground (active) window.

Error Codes

If the calling application is the active window the return code value is one, otherwise it is zero.

Example

```
C_LONGINT($lErr)

$lErr:=sys_IsAppFrontmost

If ($lErr=1)
      ALERT("We are the active window!")
Else
      ALERT("We are NOT the active window!")
End if
```

sys_IsAppLoaded

sys_IsAppLoaded(appName)→returnCode

Parameter	Type	Description
processName	text	[In] Any application with an extension of '.exe'.
returnCode	longint	[Out] 1 if app is running, 0 if not.

Description

The **sys_IsAppLoaded** function checks the list of processes currently running on the computer to determine if the application process named "processName" is running. The process must be an executable application.

Note: To see the list of all running processes in Windows NT, Windows 2000, or Windows XP, right-click on the task bar, select Task Manager from the menu, and click on the Processes tab.

Parameters

processName– text. The name of a Windows application.

Error Codes

If the function finds that the application is running, the error code return value is one, otherwise the error code return value is zero.

Example

The following code fragment tests to see if the Microsoft Calculator application is currently running. If it is not, the Win32API command **sys_ShellExecute** is used to launch it.

```
C_LONGINT($lErr;$lAppLoaded)

` Is Microsoft’s calculator accessory loaded?
$lAppLoaded:=sys_IsAppLoaded("calc.exe”)

If ($lAppLoaded=0) ` No, so run it
  $lErr:=sys_ShellExecute(“open”;”calc.exe”;””;””;SW_SHOWNORMAL)
End if
```

sys_IsAppRunningAsService

sys_IsAppRunningAsService→returnCode

Parameter	Type	Description
returnCode	longint	[Out] 1 if app is running as a service, 0 if not.

Description

The **sys_IsAppRunningAsService** function checks to see if the application is running as a Windows service.

Error Codes

If the function finds that the application is running as a service, the error code return value is one, otherwise the error code return value is zero.

Example

The following code fragment tests to see if the current application is running as a service.

```
C_LONGINT($lErr;$lAppLoaded)

` Are we currently running as a Windows service?
$lErr:=sys_IsAppRunningAsService()

If ($lErr = 0) ` No, we are not running as a service
  ALERT("This application should run as a Windows service")
End if
```

sys_IsConnectedToInternet

sys_IsConnectedToInternet→returnCode

Parameter	Type	Description
returnCode	longint	[Out] 1 if a default Internet connection has been configured, 0 if no default Internet connection has been configured.

Description

The **sys_IsConnectedToInternet** function checks to see if a default Internet connection (a "connectoid" as Microsoft calls them) has been configured. Normally, if you issue a 4D command that requires the Internet, if the autodial feature of Windows is enabled then attempting the connection may cause the Windows Internet dialup connectoid to be opened, and you will be prompted with your credentials to connect. With this function, you can test to see if a connection has been configured before you issue such a command in 4D.

NOTE: This function does NOT test to see if there is a live connection to the Internet or if a specific host is reachable. It only tests to see if the user has ever configured a default Internet connection on the computer.

Error Codes

If the function finds a configured Internet connection, the error code return value is one, otherwise the error code return value is zero.

Example

The following code fragment tests to see if user has configured a default Internet connection and presents an alert dialog.

```
C_LONGINT($lHasConnection)

` Is there a configured Internet connection?
$lHasConnection:=sys_IsConnectedToInternet

If ($lHasConnection=0)
  ALERT ("There is no Internet connection configured.")
Else
  ALERT ("This is an Internet connection configured.")
End if
```

sys_IsMultiByte

sys_IsMultiByte(byte)→errorCode

Parameter	Type	Description
byte	string	[In] Character to test.
errorCode	longint	[Out] Error code.

Description

The **sys_IsMultiByte** call determines whether a character is a lead byte, which is the first byte of a character in a double-byte character set (DBCS).

Parameters

byte – Text variable containing one character. This specifies the character to be tested. If the text variable contains more than one character, only the first character is tested.

Error Codes

If the character is a lead byte, then the error code return value is non-zero. If the character is not a lead byte, then the error code return value is zero.

Remarks

Lead bytes are unique to double-byte character sets. A lead byte introduces a double-byte character, and occupies a specific range of byte values.

The **sys_IsMultiByte** function uses the ASCII code.

Example

This example tests the ASCII value of 48 in the operating system's current language to determine if it is a lead byte.

```
C_LONGINT($lErr)
$lErr:=sys_IsMultiByte (Char(48))
If ($lErr#0)
  ` then it's a lead byte
Else
  ` then it's not a lead byte
End if
```

sys_KillProcessByID

sys_KillProcessByID(processID;cleanFirst)→errorCode

Parameter	Type	Description
processID	longint	[In] The ID of the process to kill.
cleanFirst	longint	[In] Flag to attempt clean termination first.
errorCode	longint	[Out] Error code

Description

The **sys_KillProcessByID** call terminates an external Windows process.

Parameters

processID – longint. This is the Windows PID of the process to terminate.

cleanFirst – longint. This flag determines whether or not a clean termination should be attempted first. A clean termination is analogous to clicking the Close button. If the flag is one, then it first attempts the clean termination. If it is zero, or the clean termination fails, then the process will be killed. This killing of the process is analogous to selecting the process in the Task Manager and clicking End Process.

Error Codes

If the function succeeds, the error code return value is zero. If the function fails, the error code return value will be set to the Windows error code that corresponds to the generated error.

Remarks

Note that the clean termination will happen exactly as if the Close button had been clicked in the application. This means that if the application isn't in a state that it is ready to exit (such as unsaved data, dialog box, etc.), the API could return that it was terminated when it actually wasn't. This cannot be worked around because there is no return value from a destined-to-die application that the application actually closed, just that it accepted the message to close.

Example

```
C_LONGINT($lErr;$lPos)
C_TEXT($1;$tProcName)
ARRAY TEXT(atProcNames;0)
ARRAY LONGINT(atProcIDs;0)

$lErr:=sys_EnumProcesses(atProcNames;atProcIDs)
if($lErr=0)
  $lPos:=Find in array(atProcNames;$tProcName)
  if($lPos>0)
    $lErr:=sys_KillProcessById(atProcIDs{$lPos};0)
  end if
end if
```

sys_KillProcessByName

sys_KillProcessByName(processName;mode;cleanFirst)→errorCode

Parameter	Type	Description
processName	text	[In] The name of the process to terminate.
mode	longint	[In] The mode of operation for the command.
cleanFirst	longint	[In] Flag to attempt clean termination first.
errorCode	longint	[Out] Error code.

Description

The **sys_KillProcessByName** call terminates an external Windows process.

Parameters

processName – text. This is the name process to terminate.

mode – longint. The mode determines how many processes to delete. If a one is passed for the mode, then only the first process matching the given name will be terminated. If any other integer is passed, then all of the processes matching the given name will be terminated.

cleanFirst – longint. This flag determines whether or not a clean termination should be attempted first. A clean termination is analogous to clicking the Close button. If the flag is one, then it first attempts the clean termination. If it is zero, or the clean termination fails, then the process will be killed. This killing of the process is analogous to selecting the process in the Task Manager and clicking End Process.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code is zero.

Remarks

Note that the clean termination will happen exactly as if the Close button had been clicked in the application. This means that if the application isn't in a state that it is ready to exit (such as unsaved data, dialog box, etc.), the API could return that it was terminated when it actually wasn't. This cannot be worked around because there is no return value from a destined-to-die application that the application actually closed, just that it accepted the message to close.

Example

```
C_LONGINT($lErr)

` Don't want notepad running
$lErr:=sys_KillProcessByName("notepad.exe";2;0)
if($lErr=0)
  ` Process was terminated
end if
```

sys_LogonUser

sys_LogonUser(username;domain;password)→valid

Parameter	Type	Description
username	text	[In] The username of the user to authenticate.
domain	text	[In] The domain to authenticate the user against.
password	text	[In] The password of the user to authenticate.
valid	longint	[Out] Return values.

Description

The **sys_LogonUser** command validates passed login credentials against the domain. This is useful for checking user accounts against Windows usernames and passwords.

Parameters

username – text. This parameter is the username of the user whose credentials are to be validated.

domain – text. This parameter is the domain against which the user's credentials are to be validated. If this parameter is left blank, then any domain qualification must be included in the username as domain\username. If this qualification is left off, then the username will authenticate against the local machine's account list. In addition, if this parameter is set to ".", then the username will be validated against the local machine's account list.

password – text. This parameter is the password of the user whose credentials are to be validated.

Return Values

A one is returned if the user's credentials are valid, and zero if they aren't or there was some failure in verification.

Example

```
C_BOOLEAN($bValid)
C_TEXT($1;$tUserName)
C_TEXT($2;$tDomain)
C_TEXT($3;$tPassword)

$tUserName:=$1
$tDomain:=$2
$tPassword:=$3

$bValid:=(sys_LogonUser($tUserName;$tDomain;$tPassword)=1)
```

```
if($bValid)
  ` The provided credentials are valid
  ` Perform logon related activities
else
  ` They aren't
  ` Repeat Login credentials retrieval or exit 4D
end if
```

sys_PlayWav

sys_PlayWav(fileName;flag)→errorCode

Parameter	Type	Description
fileName	text	[In] Complete path and file name of the wave file.
flag	longint	[In] Optional flag that overrides any filename and uses wave files setup for system sounds listed below.
errorCode	longint	[Out] Error code.

Description

The **sys_PlayWav** call plays a wave file specified by the filename or uses sounds established in Control Panel/Sounds and Multimedia for system events as listed below. This command performs better than 4D's PLAY command if you need to play wave files in rapid succession, such as providing a key click sound effect using 4D's On Before Keystroke form event.

Parameters

fileName – text. Complete path name and file name for the wave file (file extension .wav).

flag – longint. This variable receives a number that corresponds to the following constants. These specify a wave file setup in Control Panel for corresponding system events.

Constant	Description
MB_ASTERISK (64)	System Asterisk
MB_EXCLAMATION (48)	System Exclamation
MB_QUESTION (32)	System Question
MB_OK (0)	System Default

Error Codes

Zero is returned if the call fails. The flag value is returned when the flag is used; otherwise the return value is 1.

Example 1

```
C_TEXT($tFileName)
C_LONGINT($lErr;$lFlag)
  ` plays siren.wav
$tFileName:="C:\Temp\Siren.wav"
$lFlag:=0
$lErr:=sys_PlayWav ($tFileName;$lFlag)
```

Example 2

```
C_TEXT($tFileName)
C_LONGINT($lErr;$lFlag)
  ` plays sound assigned to system event exclamation
$lFlag:=MB_EXCLAMATION
$tFileName:=""
$lErr:=sys_PlayWav ($tFileName;$lFlag)
```

sys_PrintDirect2Driver

sys_PrintDirect2Driver(printerName;data)→errorCode

Parameter	Type	Description
printerName	text	[In] Windows name for driver to print to.
data	text	[In] Data to print to driver.
errorCode	longint	[Out] Error code.

Description

The **sys_PrintDirect2Driver** function sets outputs raw data to the specified print driver. This command has been deprecated and is kept only for legacy support. **sys_SendRawPrinterData** has replaced it for newer operating systems.

Parameters

printerName – text. This is the name of the print driver associated with Windows. Take care to make sure that you choose the correct name, as the Printers and Faxes Control Panel window does not always accurately show the name. To view the correct name, select the properties of an individual printer and view the name listed in the General tab of its properties.

data – text. This is the data to output to the printer.

Error Codes

If the function succeeds, the error code returns a value of zero. If the function fails, the error code return value is non-zero. The specific meaning of the error code can be found by looking up that value in the Windows System Error code list.

Example

This example receives the name of the default printer, and then prints data to it.

```
C_TEXT($1;$tData)
C_TEXT($tPrinterName)
C_LONGINT($lErr)
$tData:=$1
$lErr:=sys_GetDefPrinter($tPrinterName)
if($lErr=0)
  $lErr:=sys_PrintDirect2Driver($tPrinterName;$tData)
end if
```

sys_SendRawPrinterData

sys_SendRawPrinterData(printerName;data)→errorCode

Parameter	Type	Description
printerName	text	[In] Windows name for driver to print to.
data	text	[In] Data to print.
errorCode	longint	[Out] Error code.

Description

The **sys_SetSendRawPrinterData** command sends the given data directly the print spooler of the given printer. This should be used in place of **sys_printDirect2Driver** on operating systems newer than Windows XP.

Parameters

printerName – text. This parameter is the name of the print driver associated with Windows. Take care to make sure that you choose the correct name, as the Printers and Faxes Control Panel window does not always accurately show the name. To view the correct name, select the properties of an individual printer and view the name listed in the General tab of its properties. Alternatively, this can be retrieved using **sys_GetPrintJob**.

data – text. This parameter is the data to output to the printer.

Error Codes

If the function succeeds, the error code returns a value of zero. If the function fails, the error code return value is non-zero. The specific meaning of the error code can be found by looking up that value in the Windows System Error code list.

Example

This example prompts the user to choose a printer, and then prints the data to that printer.

```
ARRAY TEXT($atPrinterInfo;10)
C_TEXT($tData)
C_LONGINT($lErr)

$tData:="Command Example"

$lErr:=sys_GetPrintJob($atPrinterInfo)
If($lErr>0)
$lErr:=sys_SendRawPrinterData($atPrinterInfo{1};$tData)
End if
```

sys_SetClientDate

sys_SetClientDate(dServerCurrentDate;lForceChange)→errorCode

Parameter	Type	Description
dServerCurrentDate	4D date	[In] Current server date obtained using Current Date(*).
lForceChange	longint	[In] Any number or constant.
errorCode	longint	[Out] Error code.

Description

This function syncs the client workstation to the server date. It does nothing on 4D Server or single user.

Parameters

dServerCurrentDate – 4D date. Current server date.

lForceChange – longint. Constant determining if the date must be updated immediately.

Constant	Description
DT_FORCE_UPDATE (1)	Causes workstation date to be set to server date immediately (see Remarks).

Remarks

Workstations running on a domain server with a time provider will sync to the domain server. Unless 4D Server is running on the domain server, the date and times between 4D Client and 4D Server will not be synchronized. There may be an occasion when an immediate update to the server date/time is required and the DT_FORCE_UPDATE constant may be used as the second parameter to the function call. Note that the time will again be adjusted by the domain server on its schedule.

Error Codes

The function returns –2 if executed from 4D server or single user. Error code –1 indicates a time provider is enabled on this workstation (see remarks). It returns zero if it fails and a positive value if successful.

Example 1

```
C_LONGINT($lErr)
$lErr:=sys_SetClientDate (Current date(*))
```

Example 2 (Force workstation update regardless of time provider)

```
C_LONGINT($lErr)
$lErr:=sys_SetClientDate (Current date(*);DT_FORCE_UPDATE )
```

sys_SetClientTime

sys_SetClientTime(hServerCurrentTime;lForceChange)→errorCode

Parameter	Type	Description
hServerCurrentTime	4D time	[In] Current server time obtained using Current Time(*).
lForceChange	longint	[In] Any number or constant.
errorCode	longint	[Out] Error code.

Description

This function syncs the client workstation to the server time. It does nothing on 4D Server or single user.

Parameters

hServerCurrentTime – 4D time. Current server time.

lForceChange – longint. Constant determining if the time must be updated immediately.

Constant	Description
DT_FORCE_UPDATE (1)	Causes workstation time to be set to server date immediately (see remarks).

Error Codes

The function returns –2 if executed from 4D server or single user. Error code –1 indicates a time provider is enabled on this workstation (see remarks). It returns 0 if it fails and a positive value if successful.

Example 1

```
C_LONGINT($lErr)
$lErr:=sys_SetClientTime (Current time(*))
```

Example 2 (Force workstation update regardless of time provider)

```
C_LONGINT($lErr)
$lErr:=sys_SetClientTime (Current time(*);DT_FORCE_UPDATE )
```

sys_SetDefPrinter

sys_SetDefPrinter(printerName)→errorCode

Parameter	Type	Description
printerName	text	[In] Printer device name.
errorCode	longint	[Out] Error code.

Description

The **sys_SetDefPrinter** call establishes the current Windows default printer.

Parameters

printerName – text. This is the printer device name that you wish to use for the default printer.

Error Codes

If the function succeeds, the error code return value is non-zero.

Example

```
C_LONGINT($lErr)
C_TEXT($tNewPrinter)
$tNewPrinter:="WinFax,winspool,Ne00:"
$lErr:=sys_SetDefPrinter ($tNewPrinter)
If ($lErr>0)  ` No error
  ` continue with your code
Else
  ALERT("The printer could not be switched.")
End if
```

sys_SetEnv

sys_SetEnv(name;value)→errorCode

Parameter	Type	Description
name	text	[In] The name of the environment variable.
value	text	[In] The value of the environment variable.
errorCode	longint	[Out] Error code.

Description

The **sys_SetEnv** command assigns a value to an environment variable. An environmental variable will be created if it does not exist.

Parameters

name – text. This parameter is the name of the environment variable.

value – text. This parameter is the value to assign to the variable.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

This function only affects the environment variables of the calling process, not the global Windows variables.

sys_SetPluginLanguage

sys_SetPluginLanguage (printerName;data)→errorCode

Parameter	Type	Description
language	longint	[In] Language code
errorCode	longint	[Out] Language code.

Description

The **sys_SetPluginLanguage** command sets the language of output error messages returned by **sys_ShellExecute**. Currently English and Dutch are supported.

Parameters

language – longint. This is a constant to determine the output language to use.

Constant	Description
LANG_ENGLISH (9)	This is the default language. If this is the selected language, then the supported API error messages will be output in English.
LANG_DUTCH (19)	This code will cause the supported API error messages to be output in Dutch.

Supported Error Messages:

Invalid Operation

Invalid HowToShow Constant

File Not Found

Path Not Found

.EXE File is Invalid

OS Denied Access to File

File Name Association is Incomplete or Invalid

DDE Transaction Could Not be Completed

DDE Request Timed Out

DLL Library Not Found

No Application Associated with File Extension

Insufficient Memory

Sharing Violation Occurred

Unknown error occurred

Error Codes

The function will return the language code for the language that the plugin was set to upon completion of this command.

Example

This example sets the plugin language to Dutch

```
C_LONGINT($lErr;$lLangCode)
$lLangCode:=19
$lErr:=sys_SetPluginLanguage($lLangCode)
if($lErr=$lLangCode)
  ALERT("Plugin language set to Dutch")
else
  ALERT("Plugin Language set to English")
```

end if

sys_SetRegArray

sys_SetRegArray(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
value	text array	[Out] The value to assign to the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_SetRegArray** command queries for the specified registry entry of type array. If the entry cannot be found it will be created. The value of the entry will be set to the value passed in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – text array. This parameter is the value to assign to the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type array to work with this command.

sys_SetRegBlob

sys_SetRegBlob(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
value	blob	[Out] The value to assign to the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_SetRegBlob** command queries for the specified registry entry of type binary. If the entry cannot be found it will be created. The value of the entry will be set to the value passed in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – blob. This parameter is the value to assign to the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type binary to work with this command.

sys_SetRegLongint

sys_SetRegLongint(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being opened.
subKey	text	[In] The subkey of the registry key being opened.
name	text	[In] The name of the key being opened.
value	longint	[Out] The value to assign to the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_SetRegLongint** command queries for the specified registry entry of type longint. If the entry cannot be found it will be created. The value of the entry will be set to the value passed in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants.

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – longint. This parameter is the value to assign to the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type longint to work with this command.

sys_SetRegText

sys_GetRegText(rootKey;subKey;name;value)→errorCode

Parameter	Type	Description
rootKey	longint	[In] The root key of the registry key being set.
subKey	text	[In] The subkey of the registry key being set.
name	text	[In] The name of the key being set.
value	text	[Out] The value to assign to the registry key.
errorCode	longint	[Out] Error code.

Description

The **sys_SetRegText** command queries for the specified registry entry of type text. If the entry cannot be found it will be created. The value of the entry will be set to the value passed in the *value* parameter.

Parameters

rootKey – longint. This parameter is a 4D constant that specifies the root of the registry key. See **sys_GetRegText** for a list of constants

subKey – text. This parameter is the subkey of the registry key being queried.

name – text. This parameter is the name of the registry key.

value – text. This parameter is the value to assign to the registry key.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Remarks

Registry values must be of type text to work with this command.

Example

```
C_LONGINT($lErr)
C_TEXT($tText)

` Assign a value to the specified key.
$tText:="New Value"
$lErr:=sys_SetRegText (GR_HKEY_CURRENT_USER ;"Control
Panel\\Accessibility\\HighContrast";"High Contrast
Scheme";$tText)
```

sys_ShellExecute

sys_ShellExecute(operation;file;parameters;directory;howToShow)→errorCode

Parameter	Type	Description
operation	text	[In] May be "open", "explore", "print", or an empty string ("").
file	text	[In] Name of file.
parameters	text	[In] Parameters that should be passed to the file upon open.
directory	text	[In] Default directory to find file.
howToShow	longint	[In] Constant indicating how the file should display when opened.
errorCode	text	[Out] Error text.

Description

The **sys_ShellExecute** function opens, prints, or explores a file. In the case of an executable file, the executable is run. For a directory or folder, the folder is opened. Document files may be opened if an association for that file type exists.

Parameters

operation – text. Can be "open," "print," "explore," or an empty string (""). If an empty string is passed and a document is specified, the default operation for the associated application used.

file – text . File name or fully qualified file name.

parameters – text. Text string of parameters with each parameter separated by a space. May be an empty string.

directory – text. Fully qualified drive/path. May be an empty string if file will be found in system search path or file has an association.

howToShow – longint. Constant indicating how the executable will be displayed. ***Not used for non-executable files***. Use ONLY one of the constants below.

Constant	Description
SW_HIDE (0)	Hides the window.
SW_SHOWNORMAL (1)	Activates and displays a window. If it is minimized or maximized, Windows restores it to its original position and size. This flag should be specified when opening this window for the first time.
SW_SHOWMINIMIZED (2)	Displays the window minimized.
SW_SHOWMAXIMIZED (3)	Displays the window maximized.

SW_SHOWNOACTIVATE (4)	Displays a window in its most recent position. Current active window remains active.
SW_SHOW (5)	Activates the window and displays it in its current position.
SW_MINIMIZE (6)	Minimizes the window.
SW_SHOWMINNOACTIVE (7)	Displays the window as a minimized window. The active window remains active.
SW_SHOWNA (8)	Shows window in its current state. The active window remains active.
SW_RESTORE (9)	Activates and displays the window. If the window is minimized or maximized, it is restored to its original size and position. This flag should be used to restore a minimized window.

Error Codes

Errors in the function are indicated by a returned text string. An empty string means the function was successful. Other strings describe the cause of the error. Error codes are numerous, but the ones that you may encounter most are:

File Not Found

Path Not Found

File Name Association is Incomplete or Invalid

Sharing Violation Occurred

OS Denied Access to File

.EXE File is Invalid

Example

```
C_TEXT($tErr)
C_TEXT($tOperation;$tFile;$tParams;$tDirectory)
C_LONGINT($lHowToShow)
$tOperation:=”open”
$tFile:=”Iexplore.exe”
$tParams:=”http://www.orchardsoft.com”
$tDirectory:=””
$lHowToShow:=SW_SHOWNORMAL

$tErr:=sys_ShellExecute($tOperation;$tFile;$tParams;$tDirectory;
      $lHowToShow)
If ($tErr#””)
  ALERT("Error encountered: “+$tErr)
```

```
Else
  `Internet Explorer should be displayed
End If
```

GUI Methods

gui_DelMenuItem

gui_DelMenuItem(windowHandle;menuNum;menuItem)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
menuNum	longint	[In] Menu number.
menuItem	longint	[In] Menu item.
errorCode	longint	[Out] Error code.

Description

The **gui_DelMenuItem** function deletes a menu item from the specified menu.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

menuNum – longint. This is the menu number, starting at 0 (typically the File menu)

menuItem – longint. This is the menu item (starting at 1) from the top to delete.

***Important Note*:** 4D redraws its menu bar a few seconds after your On Startup method runs. If you notice that your calls to **gui_DelMenuItem** are not being preserved (but work when you are tracing through the debugger), add a short delay before calling this command to avoid 4D's menu bar redraw during startup.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Example

This example deletes the first three items (and the separator) from the Help menu, so the 4D help files are not shown to users. In this example, the Help menu is the 8th menu in the menu bar.

```
C_LONGINT($lWindowHandle;$lErr)
$lWindowHandle:=gui_GetWindow ("")
If ($lWindowHandle>0)
  $lErr:=gui_DelMenuItem ($lWindowHandle;7;4)
  $lErr:=gui_DelMenuItem ($lWindowHandle;7;3)
  $lErr:=gui_DelMenuItem ($lWindowHandle;7;2)
  $lErr:=gui_DelMenuItem ($lWindowHandle;7;1)
End if
```

gui_DisableCloseBox

gui_DisableCloseBox(windowHandle)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
errorCode	longint	[Out] Error code.

Description

The **gui_DisableCloseBox** call disables the Close Box.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** (see the **gui_GetWindow** section).

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Remarks

Once this call is used, there is no way to re-enable the close box.

Example

This example will disable the close box of the main 4D application window.

```
C_LONGINT($lErr;$lWindowHandle)
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_DisableCloseBox ($lWindowHandle)
End if
```

gui_FlashWindow

gui_FlashWindow(windowHandle;flags;count;rate)→errorCode

COMPATIBILITY: Requires User32.dll version 4 or later. Prior to that version, the window will only flash once. The count and tray options are not available.

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
flags	longint	[In] Action for window flash.
count	longint	[In] Number of flash repetitions.
rate	longint	[In] Flash rate in milliseconds.
errorCode	longint	[Out] Error code.

Description

The **gui_FlashWindow** flashes a window title bar.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** (see the **gui_GetWindow** section).

flags – longint.

Constant	Description
FLASHW_STOP (0)	Stops the window from flashing.
FLASHW_CAPTION (1)	Use with count of 0 to flash once, increase count the number of times to flash the caption. Use OR'd with FLASHW_TIMER(4) to flash caption until calling the function again with constant FLASHW_STOP(0).
FLASHW_TRAY (2)	Use to flash the task bar application button.
FLASHW_ALL (3)	Use to flash both the task bar application button and the window captions.
FLASHW_TIMER (4)	Use OR'd with FLASHW_ CAPTION (0) to flash caption until calling the function again with constant FLASHW_STOP(0).
FLASHW_BRING_TO_FOREGROUND (12)	Use to flash the window caption or tray window until the window is brought to the foreground.

count – longint. Number of times window caption should flash.

rate – longint. Rate of flash in milliseconds. Use zero to flash at the cursor blink rate.

Error Codes

The function returns the state of the window before it was flashed – zero if previously inactive or one if previously active.

Remarks

The count is ignored when the FLASHW_TIMER flag is used. Caption will flash until function is called with FLASHW_STOP.

Example

```
C_LONGINT($lState;$lWindow;$lFlags;$lFlashCount;$lFlashRate)
$lWindow:=gui_GetWindow ("System Information")
$lFlags:=FLASHW_CAPTION  | FLASHW_TIMER
$lFlashCount:=0
$lFlashRate:=300
$lState:=gui_FlashWindow ($lWindow;$lFlags;$lFlashCount;
      $lFlashRate)
$lFlags:=FLASHW_STOP
$lState:=gui_FlashWindow ($lWindow;$lFlags;$lFlashCount;
      $lFlashRate)
```

gui_GetDisplayFontDPI

gui_GetDisplayFontDPI(dpi)→errorCode

Parameter	Type	Description
dpi	longint	[Out] Dots per inch for display fonts.
errorCode	longint	[Out] Error code.

Description

This call retrieves the dots per inch (dpi) for small display fonts and large display fonts as defined in the Display Settings advanced window. See remarks for how to interpret these values.

Parameters

dpi – longint. This is a number in decimal format.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Remarks

While unusual, display drivers can return different drivers. The general values are: Small Fonts = 96, Large Fonts = 120.

Example

```
C_LONGINT($lErr;$lDPI)
$lErr:=gui_GetDisplayFontDPI ($lDPI)
ALERT("Display font DPI is: "+String($lDPI))
```

gui_GetOpenFileName and gui_GetSaveFileName

gui_GetOpenFileName(windowTitle;filePattern;fileDescription;startFolder;fileNameShort; fileNameFull;flags)→errorCode

gui_GetSaveFileName(windowTitle;filePattern;fileDescription;startFolder;fileNameShort; fileNameFull;flags)→errorCode

Parameter	Type	Description
windowTitle	text	[In] Text to display in title of window.
filePattern	text	[In] File name or file pattern to look for.
fileDescription	text	[In] Description of file or file pattern.
startFolder	text	[In] Folder to begin looking in.
fileNameShort	text	[In/Out] Base name of selected file.
fileNameFull	text	[Out] Full path name of selected file.
flags	longint	[In] Additional options.
errorCode	longint	[Out] Error code.

Description

The **gui_GetOpenFileName** function opens the Windows Open File dialog and allows the user to select a file to open. The **gui_GetSaveFileName** function opens the Windows Save File dialog and allows the user to select a file to save. The selected file is returned in the *fileNameShort* and *fileNameFull* fields.

No files are opened or saved by the plugin.

Parameters

windowTitle – text. This is the text that will be displayed as the caption of the File Open or File Save dialog. If this is empty, the default captions will be used.

filePattern – text. This is the string to use as the file name or file pattern to look for. You can specify a particular file, such as "system.cfg" (will limit files displayed to system.cfg only), or a wildcard pattern, such as "*.txt". Multiple patterns separated by a semi-colon may be used such as "*.txt;*.ini". If this is left empty, all files in the folder will be displayed, regardless of the text in *fileDescription*.

fileDescription – text. This is the description of the file or types of files the user is looking for. It can describe a single file, such as "System configuration file", or types of files, such as "Text Files" or "Text Files (*.txt)".

startFolder – text. This is the folder to begin the searches in. If this is left blank, the current folder will be used. If the OS is Windows 2000 or Windows 98 and no files of the type indicated in the *filePattern* exist in the current folder, the personal files folder of the current user is the initial folder.

fileNameShort – text. This holds the base name of the selected file. If the user does not select a file, this will be empty. Setting this to a file name before the call will populate the OpenFileName/SaveFileName dialog box's file name field with the suggested file name.

fileNameFull – text. This holds the full name of the selected file, including the full path. If the user does not select a file, this will be empty.

flags – longint. OR the constants to obtain multiple options. {OPTIONAL}

Constant	Description
FD_FILE_MUST_EXIST (4096)	Used only for **gui_OpenFileName**. Limits what the user may select to an existing file. Without this option, a new file may be created.
FD_CREATE_PROMPT (8192)	Displays a message box when the user enters a new file name.
FD_OVERWRITE_PROMPT (2)	Used only for **gui_SaveFileName**. Displays a message box when the user selects an already existing file.
FD_HIDE_UP_BUTTON (32768)	Hides the Up Directory button in the toolbar.
FD_HIDE_NEWDIRECTORY_BUTTON (1024)	Hides the Create New Directory button in the toolbar.
FD_HIDE_TOOLBAR (512)	Hides the toolbar.
FD_SELECT_DIRECTORY (2048)	Adds a field and Select button. The field displays the selected directory. Normally the file dialog will not return a directory only name – you must select a file. Use the Select button to return the complete path name.
FD_FILES_ONLY (4)	Limits the list to files – no directories are listed.
FD_DISABLE_EDIT_FIELD (16384)	Disables the edit field. Selections will display as uneditable in gray.

FD_DISABLE_LOOKIN_FIELD (256)
: Disables the Look-in drop down at the top of the dialog.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Remarks

It is highly recommended that variables be used to pass parameters to the plugin rather than string constants, e.g. "c:\main". See examples below.

Note that the file is NOT created. Only the name is returned. Create the file in 4D.

While the example below shows text variables, the file names are limited to 255 characters within the plugin.

Creating a new file when using the **gui_OpenFileName** call without using the limits parameter displays a prompt asking if the new file should be created. This prompt is not displayed when a file is created using the **gui_SaveFileName** call.

Example

The first example opens a dialog showing all files with extension .txt in the directory c:\Main. The Up Directory and New Directory buttons are hidden.

The second example opens a save dialog and defaults to the file name system.cfg being saved in the c:\system folder.

Example 1

```
C_LONGINT($lErr)
C_TEXT($tWindowTitle;$tFileType;$tFileDescription;$tStartFolder;
      $tFileNameShort;$tFileNameFull)
$tWindowTitle:="Open a file"
$tFileType:="*.TXT"
$tFileDescription:="Text Files "+$tFileType
$tStartFolder:="c:\main"
$lErr:=gui_GetOpenFileName($tWindowTitle;$tFileType;
      $tFileDescription;$tStartFolder;$tFileNameShort;
      $tFileNameFull;FD_HIDE_UP_BUTTON
      | FD_HIDE_NEWDIRECTORY_BUTTON )
If ($lErr#0)
  ALERT("File Selected: "+$tFileNameShort+", "+$tFileNameFull)
End if
```

Example 2

```
$tWindowTitle:="Save this file"
$tFileType:="system.cfg"
$tFileDescription:="System Configuration File"
$tStartFolder:="c:\System"
$tFileNameShort:="system.cfg"  `populates file name field

$lErr:=gui_GetSaveFileName
($tWindowTitle;$tFileType;$tFileDescription;$tStartFolder;
      $tFileNameShort;$tFileNameFull)
If ($lErr#0)
  ALERT("File Selected: "+$tFileNameFull)
End if
```

gui_GetSysColor

gui_GetSysColor(screenElement;redValue;greenValue;blueValue)→errorCode

Parameter	Type	Description
screenElement	longint	[In] Screen element whose color is to be retrieved.
redValue	longint	[Out] Red value 0-255.
greenValue	longint	[Out] Green value 0-255.
blueValue	longint	[Out] Blue value 0-255.
errorCode	longint	[Out] Error code.

Description

The **gui_GetSysColor** command retrieves the current color of the specified screen element. The RGB color components of the screen element will be returned in the *redValue*, *greenValue*, and *blueValue* parameters.

Parameters

screenElement – longint. This parameter is a 4D constant that specifies the screen element.

Constant	Description
COLOR_SCROLLBAR (0)	Scroll bar gray area.
COLOR_BACKGROUND (1)	Desktop.
COLOR_ACTIVECAPTION (2)	Active window title bar.
COLOR_INACTIVECAPTION (3)	Inactive window caption.
COLOR_MENU (4)	Menu background.
COLOR_WINDOW (5)	Window background.
COLOR_WINDOWFRAME (6)	Window frame.
COLOR_MENUTEXT (7)	Text in menus.
COLOR_WINDOWTEXT (8)	Text in windows.
COLOR_CAPTIONTEXT (9)	Text in caption, size box, and scroll bar arrow box.
COLOR_ACTIVEBORDER (10)	Active window border.

COLOR_INACTIVEBORDER (11)

Inactive window border.

COLOR_APPWORKSPACE (12)

Background color of multiple document interface (MDI) applications.

COLOR_HIGHLIGHT (13) Item(s) selected in a control

COLOR_HIGHLIGHTTEXT (14)

Text of item(s) selected in a control.

COLOR_3DFACE (15) Face color for three-dimensional display elements and for dialog box backgrounds.

COLOR_3DSHADOW (16) Shadow color for three-dimensional display elements (for edges facing away from the light source).

COLOR_GRAYTEXT (17) Grayed (disabled) text.

COLOR_BTNTEXT (18) Text on push buttons.

COLOR_INACTIVECAPTIONTEXT (19)

Color of text in an inactive caption.

COLOR_3DHIGHLIGHT (20)

Highlight color for three-dimensional display elements (for edges facing the light source).

COLOR_3DDKSHADOW (21)

Dark shadow for three-dimensional display elements.

COLOR_3DLIGHT (22) Light color for three-dimensional display elements (for edges facing the light source).

COLOR_INFOTEXT (23) Text color for tooltip controls.

COLOR_INFOBK (24) Background color for tooltip controls.

COLOR_HOTLIGHT (26) Color for a hot-tracked item. Single clicking a hot-tracked item executes the item.

COLOR_GRADIENTACTIVECAPTION (27)

Right side color in the color gradient of an active window's title bar (COLOR_ACTIVECAPTION specifies the left side color).

COLOR_GRADIENTINACTIVECAPTION (28)

Right side color in the color gradient of an inactive window's title bar (COLOR_INACTIVECAPTION specifies the left side color).

redValue – longint. This parameter is the red component color of the specified screen element.

greenValue – longint. This parameter is the green component color of the specified screen element.

blueValue – longint. This parameter is the blue component color of the specified screen element.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

Example

```
C_LONGINT($lErr)
C_LONGINT($lRVal;$lGVal;$lBVal)

` Get the color of standard Windows text.
$lErr:=gui_GetSysColor(COLOR_WINDOWTEXT;$lRVal;$lGVal;$lBVal)
```

gui_GetWindow

gui_GetWindow(windowName)→windowHandle

Parameter	Type	Description
windowName	string	[In] Window name to find.
windowHandle	longint	[Out] Window handle.

Description

The **gui_GetWindow** call retrieves the Windows Window handle for the 4D window with the corresponding window name.

Parameters

windowName – string. This is the title of a Client MDI window in the main 4D environment for which you wish to search. This must be the exact string; you may not use wildcards.

To retrieve the handle to the main 4D, 4D Client, or 4D Server window, pass the empty string ("") for *windowName*.

To retrieve the handle to the frontmost window, pass an asterisk ("*") for the *windowName*.

Error Codes

If the function succeeds, the return value is non-zero and is used as the Window handle for other GUI_ calls in this plugin.

If the function fails, the return value is zero.

Remarks

The function is tested on Windows 95/98/NT/2000/XP with 4D 6.5 and 6.7. Because of the way this function works, there is a possibility that this command could break in future versions of 4th Dimension.

The Window handle returned by this function is used by almost every other GUI function in this plugin.

Example

This example will change the name of the main 4th Dimension, 4D Client, or 4D Server application window, as well as the name that appears in the Windows Task Bar, depending on the runtime environment. This code snippet could be called in an application's On Startup method as well as in its On Server Startup method.

See the description of **gui_SetWindowTitle** for more information about this function.

```
C_LONGINT($lErr;$lWindowHandle)
C_STRING(80;$s804DAppName)
Case of
  : (Application type=4D Server )
    $s804DAppName:="CoolApp Server"
```

```
  : (Application type=4D Client )
    $s804DAppName:="CoolApp Client"
  Else
    $s804DAppName:="CoolApp"
End case
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_SetWindowTitle ($lWindowHandle;$s804DAppName)
End if
```

gui_GetWindowFrom4DWin

gui_GetWindowFrom4DWin(4DWindowNumber;serverValue)→windowHandle

Parameter	Type	Description
4DWindowNumber	longint	[In] 4D window number.
serverValue	longint	[In] Optional value to specify if the command is to be executed on the server..
windowHandle	longint	[Out] Window handle.

Description

The **gui_GetWindowFrom4DWin** call retrieves the Windows Window handle for the window with the 4D window number.

Parameters

4DWindowNumber – longint. This is a 4D window number, such as the value returned by the 4D command Frontmost Window or Open Window.

serverValue – optional. Must be 1 to execute the command on the server.

Error Codes

If the function succeeds, the return value is non-zero and is used as the Window handle for other GUI_ calls in this plugin.

If the function fails, the return value is zero.

Remarks

The Window handle returned by this function is used by every other GUI function in this plugin.

Example

This example will load an icon and place it as the icon of the frontmost window.

See the descriptions of **gui_LoadIcon** and **gui_SetIcon** for more information about these functions.

```
C_LONGINT($lErr;$l4DWinNumber$lWindowHandle;$hIcon)
$l4DWinNumber:=Frontmost window
$lWindowHandle:=gui_GetWindowFrom4DWin ($l4DWinNumber)
If ($lWindowHandle>0)
  $lErr:=gui_LoadIcon ("c:\main.ico";$hIcon)
  If ($hIcon>0)
    $lErr:=gui_SetIcon ($lWindowHandle;$hIcon)
  End if
End if
```

Example 2

This example will execute the command on the server. It will load an icon and place it as the icon of the frontmost window.

See the descriptions of **gui_LoadIcon** and **gui_SetIcon** for more information.

```
C_LONGINT($lErr;$l4DWinNumber$lWindowHandle;$hIcon)
$l4DWinNumber:=Frontmost window
$lWindowHandle:=gui_GetWindowFrom4DWin ($l4DWinNumber;1)
If ($lWindowHandle>0)
  $lErr:=gui_LoadIcon ("c:\main.ico";$hIcon)
  If ($hIcon>0)
    $lErr:=gui_SetIcon ($lWindowHandle;$hIcon)
  End if
End if
```

gui_GetWindowState

gui_GetWindowState(windowHandle)→state

Parameter	Type	Description
windowHandle	longint	[In] Window handle.
state	longint	[Out] Constant indicating the window state.

Description

The **gui_GetWindowState** retrieves the state (minimized, maximized, or normal) of the window.

Parameters

windowHandle – longint. Window handle returned by **gui_getWindow** or **gui_getWindowFrom4DWin**.

Error Codes

The function returns zero for a window that is neither minimized nor maximized. The following constants are returned when the window is minimized or maximized.

Constant	Description
IS_MINIMIZED (1)	Window is minimized.
IS_MAXIMIZED (2)	Window is maximized.

Example

```
C_LONGINT($lState;$lWindowHandle)
$lWindowHandle:=gui_GetWindow ("MyWindow")
$lState:=gui_GetWindowState ($lWindowHandle)
If ($lState=IS_MINIMIZED )
  `do whatever
End if
```

gui_GetWindowStyle

gui_GetWindowStyle(windowHandle;styleList)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle.
styleList	array text	[Out] Window styles.
errorCode	longint	[Out] Error code.

Description

The **gui_GetWindowStyle** command retrieves the styles for the window indicated by the window handle.

Parameters

windowHandle – longint. This is the window handle retrieved using **gui_GetWindow**.

styleList -- text array. This is the text array variable initialized to zero elements. It is used to receive the list of style attributes for the selected window. The array is filled with text representations of the constants used by Windows (and also defined as Win32API constants).

Text	Related Win32API Constant
"WS_OVERLAPPED"	WS_OVERLAPPED (0)
"WS_MAXIMIZEBOX"	WS_MAXIMIZEBOX (65536)
"WS_MINIMIZEBOX"	WS_MINIMIZEBOX (131072)
"WS_THICKFRAME"	WS_THICKFRAME (262144)
"WS_SYSMENU"	WS_SYSMENU (524288)
"WS_HSCROLL"	WS_HSCROLL (1048576)
"WS_VSCROLL"	WS_VSCROLL (2097152)
"WS_DLGFRAME"	WS_DLGFRAME (4191304)
"WS_BORDER"	WS_BORDER (8388608)
"WS_CAPTION"	WS_CAPTION (12585912)
"WS_CLIPCHILDREN"	WS_CLIPCHILDREN (33554432)
"WS_CLIPSIBLINGS"	WS_CLIPSIBLINGS (67108864)
"WS_DISABLED"	WS_DISABLED (134217728)
"WS_VISIBLE"	WS_VISIBLE (268435456)
"WS_CHILD"	WS_CHILD (1073741824)
"WS_POPUP"	WS_POPUP (2147483648)

Error Codes

The function returns non-zero if successful and zero if it fails.

Example

```
C_LONGINT($lErr;$lWindowHandle)
ARRAY TEXT($atStyleList;0)

$lWindowHandle:=gui_GetWindow ("MyWindow")
$lState:=gui_GetWindowStyle ($lWindowHandle;$atStyleList)
If (Find in array($atStyleList;"WS_THICKFRAME")=-1)
  `window has a sizing border
End if
```

gui_GetWndRect

gui_GetWndRect(windowHandle;x;y;w;h;mode)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
x	longint	[Out] X location.
y	longint	[Out] Y location.
w	longint	[Out] Window width.
h	longint	[Out] Window length.
mode	longint	[In] Mode of functionality.
errorCode	longint	[Out] Error code.

Description

The **gui_GetWndRect** call retrieves the dimensions of the bounding rectangle of the specified window. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen. The definition of screen will depend on the value passed for the mode parameter.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

x – longint. Specifies the x-coordinate of the window: distance from the left side of the screen to the outer edge of the window.

y – longint. Specifies the y-coordinate of the window: distance from the top of the screen to the outer edge of the window.

w – longint. Specifies the width of the window: distance from outer left side to outer right side.

h – longint. Specifies the height of the window: distance from the outer top to the outer bottom.

mode – longint. Specifies which mode to operate this function in. If one is sent, then the original functionality of returning the coordinates relative to the upper-left corner of the virtual desktop. If any other number is sent, then the new functionality of returning the coordinates relative to the upper-left corner of the monitor that the window is residing in will be used.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Example

```
C_LONGINT($lErr;$lWindowHandle;$x;$y;$w;$h)
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_GetWndRect ($lWindowHandle;$x;$y;$w;$h)
End if
```

gui_HideTaskBar

gui_HideTaskBar→returnCode

Parameter	Type	Description
returnCode	longint	[Out] Return code.

Description

The **gui_HideTaskBar** function will hide the Windows task bar until a call to gui_ShowTaskBar is made.

Error Codes

If the task bar was successfully hidden the return code will be a nonzero value. If the task bar was already hidden the return code will be zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_HideTaskBar

  ` The task bar will now remain hidden
  ` until the following call to gui_ShowTaskBar

$lErr:=gui_ShowTaskBar
```

gui_HideTitleBar

gui_HideTitleBar→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_HideTitleBar** function hides the Windows title bar of the calling application.

Error Codes

If the title bar was successfully hidden the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_HideTitleBar

  ` The title bar will now remain hidden
  ` until the following call to gui_ShowTitleBar

$lErr:=gui_ShowTitleBar
```

gui_LoadBackground

gui_LoadBackground(fileName;style)→errorCode

Parameter	Type	Description
fileName	text	[In] Complete file and path of bitmap file.
style	longint	[In] Constant specifying whether the bitmap should be tiled or scaled.
errorCode	longint	[Out] Error code.

Description

The **gui_LoadBackground** call lets you set a custom bitmap image as the background for your main 4D application window. The bitmap image can be loaded or cleared at any time, and it can be tiled, scaled to fit the maximum available space (i.e. the size of the desktop), or tiled to fit the exact area of the 4D main application window.

Any Windows BMP file may be specified.

Parameters

filename – text. This parameter is either a text string or variable that specifies the complete path to the bitmap image. If you pass an empty string (“”), or the constant BM_CLOSE(“”), any existing background image will be cleared and the standard Windows background will be redrawn.

style – longint. This parameter is a 4D constant that specifies whether the image should be tiled or scaled to fit the available space.

Constant	Description
BM_TILE (1)	Tiles the image.
BM_SCALE (2)	Scales the image to the size of 4D’s main application window. The image will be rescaled if the main 4D window is resized (see the discussion below).
BM_SCALETOMAXCLIENT (3)	Scales the image to the maximized size of 4D’s main application window. If the main 4D window is resized, the image is cropped or revealed as the size of the window is decreased or increased.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Remarks

The bitmap image is held in the main application memory space for 4D or 4D Client. If you specify a very large bitmap image, you should make sure that you have enough kernel memory allocated to 4D or 4D Client. You can increase the amount of kernel memory allocated to your 4D applications using the 4D Customizer Plus application. The memory is released when 4D exits, or when you call:

```
gui_LoadBackground(BM_CLOSE)
```

If you specify BM_SCALE, the plugin will scale your bitmap image every time you resize the application window. The time required to scale the image could be noticeable, especially on slower computers. BM_SCALETOMAXCLIENT is much faster since the scaling computation only needs to take place once when the image is loaded (the image is scaled to the maximum size possible for the 4D window), and any subsequent resizing of the application window does not require extra scaling.

In versions prior to 3.6, if you called **gui_LoadBackground** with the parameter BM_SCALE, Win32API would disable live window dragging and resizing until 4D exited. This was for performance reasons only, since the Windows bitmap scaling routine was quite slow, even on fast computers. If 4D did not exit normally, the setting for live window resizing was disabled until Windows was restarted—the change was made for the current session only, not permanently.

If you choose to scale your background image, you should strongly consider using BM_SCALETOMAXCLIENT., since most people seem to run 4D using the entire area of the screen. BM_SCALETOMAXCLIENT is much faster, and it does not change any system options set by the user.

Example 1

The following code will load the bitmap image named "Background.BMP" from the Windows TEMP directory and tile it on the application background.

```
C_TEXT($tFileName)
C_LONGINT($lErr)
$tFileName:=Temporary folder+"Background.BMP"
$lErr:=gui_LoadBackground ($tFileName;BM_TILE )
```

Example 2

The following code will clear the current bitmap image set by a previous call to **gui_LoadBackground**. If there is no background set, this command does nothing.

```
C_LONGINT($lErr)
$lErr:=gui_LoadBackground (BM_CLOSE )
```

gui_LoadIcon

gui_LoadIcon(iconFileName;hIcon)→errorCode

Parameter	Type	Description
iconFileName	string	[In] Full path name of icon file to load.
hIcon	longint	[Out] Icon handle.
errorCode	longint	[Out] Error code.

Description

The **gui_LoadIcon** reads an icon file from disk and stores it in a Windows HICON handle to be used by **gui_SetIcon**.

Parameters

iconFileName – Text variable containing the full name of the icon file on the hard drive.

hIcon – Numeric handle of the icon file in memory.

Error Codes

If the file does not exist or is an invalid icon file, then the error code return value is zero. If the function succeeds, the *hIcon* variable is filled and the error code is non-zero.

Remarks

Once the icon is loaded and has been assigned a variable, the handle to the icon is valid for the life of the application. This way, icons can be loaded once at startup and used for the duration of the application.

Use the return value from this function in **gui_SetIcon**.

TIP: If you want to store your Windows icons inside your 4D structure file, encode the .ICO file into a text document using a format such as UUENCODE, and put this text into a TEXT resource in the .RSR file (write your own utility in 4D using the Resource commands, or transport your structure to a Macintosh and use ResEdit). When your application starts up, write the contents of the TEXT resource into a document in the user's temporary directory, UUDECODE the file, load the icon into memory with **gui_LoadIcon**, and then delete all your files from the temporary directory.

Example

This example will load an icon and place it as the icon of the frontmost window.

See the descriptions of **gui_LoadIcon** and **gui_SetIcon** for more information about these functions.

```
C_LONGINT($lErr;$l4DWinNumber$lWindowHandle;$hIcon)
$l4DWinNumber:=Frontmost window
$lWindowHandle:=gui_GetWindowFrom4DWin ($l4DWinNumber)
If ($lWindowHandle>0)
```

```
  $lErr:=gui_LoadIcon ("c:\main.ico";$hIcon)
  If ($hIcon>0)
    $lErr:=gui_SetIcon ($lWindowHandle;$hIcon)
  End if
End if
```

gui_MaximizeMDI

gui_MaximizeMDI→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_MaximizeMDI** function maximizes the main 4D MDI window.

Error Codes

If the window was successfully maximized the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_MaximizeMDI


If ($lErr#0 )
        ` The window is now maximized.
End if
```

gui_MinimizeMDI

gui_MinimizeMDI→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_MinimizeMDI** function minimizes the main 4D MDI window.

Error Codes

If the window was successfully minimized the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_MinimizeMDI


If ($lErr#0 )
        ` The window is now minimized.
End if
```

gui_MessageBox

gui_MessageBox(windowHandle;messageText;windowTitle;dialogType)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle.
messageText	text	[In] Message to display.
windowTitle	text	[In] Title for message box.
dialogType	longint	[In] Type of Windows message box to display.
errorCode	longint	[Out] Error code.

Description

The **gui_MessageBox** function displays a standard Windows message box with the specified title, contents and type.

Parameters

windowHandle – longint.. Window handle returned by gui_getWindow or gui_GetWidowFrom4DWin

messageText - text. The text that should be displayed in the message box.

windowTitle - text. The title for the message box.

dialogType - longint. Constants specifying the Windows dialog type to use for the message box. Multiple options can be combined with a logical OR (|).

Constant	Description
MB_OKCANCEL (1)	This message box contains two buttons, OK and Cancel.
MB_ABORTRETRYIGNORE (2)	This message box contains three buttons, Abort, Retry, and Ignore.
MB_YESNOCANCEL (3)	This message box contains three buttons, Yes, No, and Cancel.
MB_YESNO (4)	This message box contains two buttons, Yes and No.
MB_RETRYCANCEL (5)	This message box contains two buttons, Retry and Cancel.
MB_CANCELTRYCONTINUE (6)	This message box contains three buttons, Cancel, Try Again, and Continue
MB_ICONSTOP (16)	Display a stop-sign icon in the message box.
MB_ICONQUESTION (32)	Display a question-mark icon in the message box.

MB_ICONWARNING (48)	Display an exclamation-point icon in the message box.
MB_ICONINFORMATION (64)	Display an information icon (lower case 'i' in a circle) in the message box.
MB_DEFBUTTON1 (0)	This is the default option. It sets the first button on the message box as the default button.
MB_DEFBUTTON2 (256)	Set the second button on the message box as the default button.
MB_DEFBUTTON3 (512)	Set the third button on the message box as the default button.
MB_DEFBUTTON4 (768)	Set the fourth button on the message box as the default button.
MB_APPLMODAL (0)	This is the default option. Open the message box as a modal dialog within the application.
MB_SYSTEMMODAL (4096)	Open the message box as a modal dialog for the entire system.
MB_TASKMODAL (8192)	Open the message box as a modal dialog within the application. This option does not require a valid window handle to be passed in.

Error Codes

If the function fails the error code return value will be zero, otherwise it will be one of the following constants.

Constant	Description
MB_IDOK (1)	User clicked the OK button.
MB_IDCANCEL (2)	User clicked the Cancel button.
MB_IDABORT (3)	User clicked the Abort button.
MB_IDRETRY (4)	User clicked the Retry button.
MB_IDIGNORE (5)	User clicked the Ignore button.
MB_IDYES (6)	User clicked the Yes button.
MB_IDNO (7)	User clicked the No button.
MB_IDTRYAGAIN (10)	User clicked the Try Again button.
MB_IDCONTINUE (11)	User clicked the Continue button.

Example

```
C_LONGINT($lErr;$lHWnd)

$lHWnd:=gui_GetWindow

$lErr:=gui_MessageBox ($lHWnd;"Would you like to
continue?";"Alert";MB_CANCELTRYCONTINUE  | MB_ICONQUESTION )

If ($lErr=MB_IDCONTINUE )
        ` Keep processing.
Else
        ` Abort
End if
```

gui_RespectToolbar

gui_RespectToolbar(pixels;position{;extraPixels}) --> errorCode

Parameter	Type	Description
pixels	longint	[In] Pixel height of the tool bar.
position	text	[In] Code to indicate the position of the toolbar. "L" Left tool bar. "R" Right tool bar. "T" Top tool bar. "B" Bottom tool bar.
extra pixels	longint	[In] Adjustment to hide the title bar beneath the tool bar when maximized. This is valid for tool bars positioned at the top of the screen only.
errorCode	longint	[Out] Error code.

Description

The **gui_RespectToolbar** command intercepts mouse tracking and window resize messages inside the 4th Dimension MDI window to accommodate a custom tool bar window. If you use this command, you can create a tool bar window that appears on any edge of the MDI window, and when the window is maximized, the content of the window will not be obscured by the tool bar.

In addition, this command will prevent the mouse from dragging into the region defined for the toolbar, as if the tool bar actually defined the edge of the active area of the MDI window.

Parameters

pixels – longint. This is the number of pixels that a maximized window will be moved in order to prevent the content area of the window from being obscured by the tool bar. This is normally the width or the height of your tool bar form minus the size of the window caption.

Position – This is a text literal to indicate the position of the tool bar on the screen:

"T" = top

"L" = left

"R" = right

"B" = bottom

extraPixels – This is an extra adjustment that you may want for maximized windows to hide their title bar completely beneath a tool bar that appears at the top of the screen.

errorCode – This will be zero if the command failed, non-zero otherwise.

Important Notes

1. You must call **gui_SubClassInit** (RW_SUBCLASS_INIT) one time in your application, prior to opening the window that contains the tool bar. You may call this command in your application's On Startup method, if you wish.

2. You may have multiple tool bars running at the same time. Each tool bar must be in its own process, however.

3. 4D has numerous bugs in its window management routines on the Windows platform, and they show no signs of being fixed. In order to prevent these bugs from interfering with tool bars, we decided to always restore a maximized window when opening new windows or switching between windows, instead of trying to preserve the maximized state of all MDI child windows at all times. If 4D ever fixes their bugs, we may stop doing this, but for now it seems like the best approach.

4. This command fixes a long-standing 4D bug where a maximized child window inside a maximized MDI window would not be sized correctly, instead leaving a border of several pixels at the right edge of the window. Since the plugin is intercepting redraw messages, we are able to size the window correctly, thus fixing 4D's bug. If you wish to have the benefit of this bug fix but you do not have a tool bar, create a "top" tool bar of height -1 pixels from your On Startup database method, as follows:

```
$lErr:=gui_SubClassInit (RW_SUBCLASS_INIT )
$lErr:=gui_RespectToolbar (-1;"T")
```

5. It is useful for tool bars on the bottom and on the right to know when the MDI window is resized so they can be moved accordingly. A special interprocess array can be defined to receive Outside Call form events when the MDI window is resized.

When the MDI window is resized, an On Outside Call form event will be sent to each process that registered a tool bar with **gui_RespectToolbar**.

When using 4Dv11 the <>TB_NOTIFICATION array can no longer be accessed by WIN32API. This means that we will be unable to determine what toolbar triggered the On Outside Call event.

Your tool bar's form method should check this special interprocess longint array, named <>TB_NOTIFICATION, for a non-zero element in the correct place within the array:

- If the tool bar process is for a "left" tool bar, it should check <>TB_NOTIFICATION{1} for a non-zero value.

- If the tool bar process is for a "top" tool bar, it should check <>TB_NOTIFICATION{2} for a non-zero value.

- If the tool bar process is for a "right" tool bar, it should check <>TB_NOTIFICATION{3} for a non-zero value.

- If the tool bar process is for a "bottom" tool bar, it should check <>TB_NOTIFICATION{4} for a non-zero value.

After your form method has read a non-zero value from the correct position, it should set the value back to zero and then run any code required to handle a resize of the MDI window.

Example 1 - Tool bar process initialization

```
  ` This is a sample to show how to use the Win32API
  ` gui_RestrictToolbar command.

C_LONGINT($x;$lTop;$lErr;$lCaptionHeight;$lToolBarHeight)
C_BOOLEAN($bHideTitleBarWhenMaximized)

MENU BAR(1)
MESSAGES OFF

  ` When a resize of the MDI window occurs, the plugin will send
  ` an outside call form event to the processes that have called
  ` gui_RespectToolbar.

  ` Since multiple processes may be toolbars, an array is
  ` returned with elements set to non-zero for each process to
  ` check and then clear:

  ` Element 1: Left toolbar
  ` Element 2: Top toolbar
  ` Element 3: Right toolbar
  ` Element 4: Bottom toolbar

  ` This is a "top" toolbar, so we'll be sure to initialize
  ` the second element of <>TB_NOTIFICATION to zero
  ` before we begin.

ARRAY LONGINT(<>TB_NOTIFICATION;4)
<>TB_NOTIFICATION{2}:=0

  ` This variable can be used to control whether you want to see
  ` the window title bar when maximized, immediately below the
  ` toolbar, or if you want the window title bar hidden.

$bHideTitleBarWhenMaximized:=True

$lToolBarHeight:=33  ` height in pixels of the toolbar dialog
$lCaptionHeight:=sys_GetWindowMetrics (WM_CAPTION_HEIGHT )

  ` Height in pixels of a normal window title bar

$lErr:=gui_SubClassInit (RW_SUBCLASS_INIT )

  ` Enable intercepting of Windows messages
  ` Set the number of pixels to adjust 4D's vertical resizing

If ($bHideTitleBarWhenMaximized)
```

```
  $lErr:=gui_RespectToolBar ($lToolBarHeight-
$lCaptionHeight;"T";$lToolBarHeight)
Else
  $lErr:=gui_RespectToolBar
($lToolBarHeight+(sys_GetWindowMetrics (WM_BORDER_HEIGHT
)*2);"T")

  ` Palette windows are one pixel smaller than a standard caption

End if

  ` Find the top coordinate of the toolbar using Win32 calls to
  ` take into account the OS and visual theme currently in use

$lTop:=sys_GetWindowMetrics (WM_BORDER_HEIGHT
)*3+sys_GetWindowMetrics (WM_MENU_HEIGHT )-3

  ` Open the toolbar

$x:=Open window(0;$lTop;1800;$lTop+$lToolBarHeight;-1*Palette
window )
DIALOG([dialogs];"Toolbar")
CLOSE WINDOW
```

Example 2 - Outside Call handler for a "top" tool bar

```
  ` This form method belongs to the toolbar form that
  ` stretches across the top of the MDI window. It receives
  ` notifications that the MDI window has been resized.

Case of
  : (Form event=On Outside Call )

    If (Size of array(<>TB_NOTIFICATION)>=4)

        ` This is a "top" toolbar, so we need to check
        ` position 2 of the array (1=l, 2=t, 3=r, 4=b)
      If (<>TB_NOTIFICATION{2}>0)  ` Top toolbar
          ` do something here if you need to,
          ` probably more useful than beeping!
        BEEP
        <>TB_NOTIFICATION{2}:=0
      End if

    End if

End case
```

gui_RestoreMDI

gui_RestoreMDI→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_RestoreMDI** function restores the main 4D MDI window to it's un-minimized, un-maximized state.

Error Codes

If the window was successfully restored the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_RestoreMDI


If ($lErr#0 )
        ` The window is now restored.
End if
```

gui_RestrictWindow

gui_RestrictWindow(windowHandle;restriction)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle.
restriction	longint	[In] Constraint on window.
errorCode	longint	[Out] Error code.

Description

The **gui_RestrictWindow** function restricts one or more window attributes. This function targets 4D child windows – windows inside the main 4D window. Attributes that can be restricted are: sizing, minimizing, maximizing, and moving the window.

IMPORTANT: A call to **gui_SubClassInit**(RW_SUBCLASS_INIT) must be made **prior to the target window being created**. This subclasses all child windows and allows the interception of commands that pertain to changing the window behavior. When you are certain that no more calls to **gui_RestrictWindow** will be made during an application session, a call to **gui_SubClassInit**(RW_RELEASE) may be made to release the overhead of the subclassed procedure. The subclassed procedure is automatically released when 4D exits.

Constants used for **gui_SubClassInit** are:

Constant	Description
RW_SUBCLASS_INIT (1024)	Initializes the subclass.
RW_RELEASE (0)	Releases the subclass information.

Parameters

windowHandle – longint. This is the window handle retrieved using **gui_GetWindow**.

restriction -- Use one or more of the following constants to restrict the window. Multiple restrictions should be OR'd together.

Constant	Description
RW_NO_SIZE (1)	Disables resizing.
RW_NO_MOVE (2)	Disables moving.
RW_NO_MIN (4)	Disables minimizing.
RW_NO_MAX (8)	Disables maximizing.
RW_NO_NEXT (16)	Disables the "Next Window" menu option.
RW_NO_CLOSE (32)	Disables the "Close Window" menu option.

Remarks

For RW_NO_SIZE, the cursor does not change to a sizing cursor on the border or corners of the window. The size menu item on the window's system menu is also disabled. For RW_NO_MOVE, the window may not be moved using the caption area or the system menu. For RW_NO_MIN and RW_NO_MAX, the minimize and maximize buttons in the caption area and the items on the system menu are disabled.

Error Codes

The function returns zero if the function fails. It returns –1 if the window handle and restriction cannot be saved in memory. It returns a non-zero value if the function succeeds. The positive value is the number of windows currently being restricted by the subclassed procedure.

Example

```
C_LONGINT($lErr;$lWindowHandle;$lRestrictions)
$lErr:=gui_SubClassInit (RW_SUBCLASS_INIT )
If ($lErr#0)
  $lRestrictions:=RW_NO_SIZE  ^| RW_NO_MOVE
  $lWindowHandle:=gui_GetWindow ("MyWindow")
  $lErr:=gui_RestrictWindow ($lWindowHandle;$lRestrictions)
  `window can't be moved or sized
End if
```

gui_SelectColor

gui_SelectColor(redValue;greenValue;blueValue {;hasCustomColors {;customColorValues}})→errorCode

Parameter	Type	Description
redValue	longint	[In/Out] Red value 0 – 255.
greenValue	longint	[In/Out] Green value 0-255.
blueValue	longint	[In/Out] Blue value 0 – 255.
hasCustomColors	longint	[In; Optional] A value of one indicates that the customColorValues array will be passed as a parameter to the function. A value of zero, while redundant, means that the customColorValues array will not be passed to the function.
customColorValues	longint array	[In/Out; Optional] An array of up to 16 long integers representing the packed RGB values for the 16 custom colors allowed in the dialog. See the Description below for information on how to pack the data.
errorCode	longint	[Out] 0 if the user clicked the Cancel button in the dialog, non-zero if the user clicked the OK button.

Description

The **gui_SelectColor** function displays the Microsoft Windows color picker common dialog box in a movable, modal dialog window.

Set the values for *redValue*, *greenValue*, and *blueValue* parameters before calling the function to display the dialog with a default color pre-selected. These numbers must be between 0 and 255. If all three variables are 0, then black will be pre-selected. If all three variables are 255, then white will be pre-selected.

If the user clicks OK in the Color dialog, *errorCode* will return a non-zero value; *redValue*, *greenValue*, and *blueValue* will contain the new selected red, green, and blue values.

If the user clicks the Cancel button in the Color dialog, *errorCode* will return 0; *redValue*, *greenValue*, and *blueValue* will all be set to zero.

If the user defines some custom colors in the Color dialog, the custom colors will appear the next time the dialog is opened in the same 4D session. The custom colors are lost when 4D exits.

You may control the custom colors area of the dialog by pre-selecting colors to display, and by reading the user's selection of custom colors for storage in your database.

The optional *hasCustomColors* and *customColorValues* parameters must be used together. If you wish to utilize the custom colors portion of the dialog, set the *hasCustomColors* flag to 1, and pass an array of between 1 and 16 longint elements in the *customColorValues* parameter. If the user clicks OK in the dialog, the array will be resized to 16 elements, and each element will contain the custom color defined in the dialog.

Each array element is a packed representation of the red, green, and blue values. In 4D code, this is calculated as follows:

```
$packedValue:=($blueValue<<16)+($greenValue<<8)+$redValue
```

Note that this is the opposite of how 4D's SET RGB COLORS command packs the values. In 4D, this would be represented as:

```
$packedValue:=($redValue<<16)+($greenValue<<8)+$blueValue
```

Important: Due to the design of the 4D plugin architecture, all 4D processes, including the built-in web server and other background processes, are suspended while the Color dialog is displayed. Avoid displaying the color picker dialog for extended periods of time on machines that are running important background processes such as web servers.

Error Code

The function returns zero if the dialog is cancelled. It returns a non-zero value if the user selects OK from the dialog.

Example

In HTML tags, RGB colors are represented as follows:

```
#RRGGBB
```

RR is the hexadecimal value for the red color, GG is the hexadecimal value for the green color, and BB is the hexadecimal value for the blue color.

For example, a dark red would be described by the following string: "#93000B". Black would be described by: "#000000". White would be described by "#FFFFFF".

In this example, we will construct a nice-looking, generic color picker on a 4D form. The color picker is simply a rectangle covered by an invisible button.

Our wrapper method for the Windows color picker, which will be called by the On Click handler of the invisible button, will return an HTML-style RGB string that could be easily stored in the database. It will also set the color of our rectangle based on the user's selection inside the Color dialog.

First, let's create a short helper method named util_numHexToDec that will convert a hexadecimal string to its decimal value:

```
` $1 = hex string
` $0 = longint

C_TEXT($1;$hexstr)
```

```
C_LONGINT($0;$result;$i)
$result:=0
$hexstr:="0123456789ABCDEF"
For ($i;Length($1);1;-1)
  $result:=$result+((Position($1[[$i]];$hexstr)-
                    1)*(16^(Length($1)-$i)))
End for
$0:=$result
```

Now, we'll create our highly generic wrapper method named util_selectHTMLColor:

```
` Method: util_selectHTMLColor

` This method returns a new RGB color selected by the user
` It presents the Windows color picker with a single custom
` color set to the color defined by $1.

` If the user cancels the dialog, the original color is
` returned. If the user selects a new color, the color
` string for the new color is returned.

` All colors are denoted in HTML style - i.e. #RRGGBB where
` RR is the hex representation of the red value, GG is the hex
` representation of the green value, and BB is the hex
` representation of the blue value.

  ` If a second parameter is passed, it should be the object name
  ` of a valid 4D form object. This method will set the color of
  ` this object to the new color selected in the color picker.

C_TEXT($0;$tSelectedColor)
C_TEXT($1;$tOriginalColor)
C_TEXT($2;$tObjectName)

$tOriginalColor:=$1
$tSelectedColor:=$tOriginalColor
If (Count parameters=2)
  $tObjectName:=$2
Else
  $tObjectName:=""
End if

If (Length($tOriginalColor)=7)
  If ($tOriginalColor[[1]]="#")
    C_LONGINT($lRed;$lGreen;$lBlue;$lErr)
      ` Parse the #RRGGBB string and extract the longint
      ` values for red, green, and blue.
    $lRed:=util_numHexToDec (Substring($tOriginalColor;2;2))
    $lGreen:=util_numHexToDec (Substring($tOriginalColor;4;2))
```

```
    $lBlue:=util_numHexToDec (Substring($tOriginalColor;6;2))
      ` Put the original color of $1 into the first square
      ` reserved for Custom Colors in the dialog.
    ARRAY LONGINT($alCustomColors;1)
    $alCustomColors{1}:=($lBlue << 16)+($lGreen << 8)+$lRed
      ` Present the Windows common dialog for color selection
    $lErr:=gui_SelectColor ($lRed;$lGreen;$lBlue;
                            1;$alCustomColors)
    If ($lErr#0)  ` If the user clicked OK…
        ` Create a new color string
      $tSelectedColor:=Replace string(String($lRed;"&x")+
                          String($lGreen;"&x")+
                          String($lBlue;"&x");"0x";"")
      $tSelectedColor:="#"+Substring($tSelectedColor;3;2)+
                          Substring($tSelectedColor;7;2)+
                          Substring($tSelectedColor;11;2)
    End if
  Else ` bad color string was passed; return a valid string
    $tSelectedColor:="#000000" ` black
  End if
Else   ` bad color string was passed - return a valid string
  $tSelectedColor:="#000000" ` black
End if

If ($tObjectName#"")
  $lRed:=util_numHexToDec (Substring($tSelectedColor;2;2))
  $lGreen:=util_numHexToDec (Substring($tSelectedColor;4;2))
  $lBlue:=util_numHexToDec (Substring($tSelectedColor;6;2))
  SET RGB COLORS(*;$tObjectName;0;($lRed << 16)+
                 ($lGreen << 8)+$lBlue)
End if

$0:=$tSelectedColor
```

Finally, to create the color picker object on any 4D form, simply follow these steps:

1. Create a small rectangle on the form. Set the object name to "rectangle1".
2. Create an invisible button or a highlight button the same size as rectangle1, and place it on top of rectangle1.
3. Create the following object method for the invisible button:

```
Case of
  : (Form event=On Clicked )
    C_TEXT($tColor)
    $tColor:=util_selectHTMLColor ($tColor;"rectangle1")
End case
```

gui_ServerUnloadBackground

gui_ServerUnloadBackground() → errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

Executing the command **gui_LoadBackground** on the 4D server could cause it to crash on shutdown. This command should be added to the On Server Shutdown database method when using **gui_LoadBackground** on the server to prevent crashing from occurring.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

gui_SetIcon

gui_SetIcon(windowHandle;hIcon)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
hIcon	longint	[In] Icon handle.
errorCode	longint	[Out] Error code.

Description

The **gui_SetIcon** replaces the icon of the specified window with the icon stored in *hIcon*.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

hIcon – Numeric handle of the icon file in memory.

Error Codes

If the function succeeds, the *hIcon* variable is filled and the error code is non-zero. If the file does not exist or is an invalid icon file, then the error code return value is zero.

Remarks

Use **gui_SetIcon** to fill the *hIcon* parameter.

TIP: 4D Insider may be used to easily create a wrapper for the 4D Open Window command. Use your wrapper method, **gui_GetWindowFrom4Dwin**, and **gui_SetIcon** to give all your custom windows a unique icon.

Example

This example will load an icon and place it as the icon of the frontmost window.

See the descriptions of **gui_LoadIcon** and **gui_SetIcon** for more information about these functions.

```
C_LONGINT($lErr;$l4DWinNumber$lWindowHandle;$hIcon)
$l4DWinNumber:=Frontmost window
$lWindowHandle:=gui_GetWindowFrom4DWin ($l4DWinNumber)
If ($lWindowHandle>0)
  $lErr:=gui_LoadIcon ("c:\main.ico";$hIcon)
  If ($hIcon>0)
    $lErr:=gui_SetIcon ($lWindowHandle;$hIcon)
  End if
End if
```

gui_SetMDIOpaque

gui_SetMDIOpaque→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_SetMDIOpaque** function will set the main 4D MDI window back to an opaque state if it was previously made transparent.

Error Codes

If the window was successfully made opaque the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_SetMDIOpaque


If ($lErr#0 )
        ` You can't see through the window anymore.
End if
```

gui_SetMDITransparent

gui_SetMDITransparent→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_SetMDITransparent** function will make the main 4D MDI window background transparent.

Error Codes

If the window was successfully made transparent the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_SetMDITransparent


If ($lErr#0 )
        ` You can see straight through the window now.
End if
```

gui_SetSysColor

gui_SetSysColor(screenElement;redValue;greenValue;blueValue)→errorCode

Parameter	Type	Description
screenElement	longint	[In] Screen element whose color is to be retrieved.
redValue	longint	[In] Red value 0-255.
greenValue	longint	[In] Green value 0-255.
blueValue	longint	[In] Blue value 0-255.
errorCode	longint	[Out] Error code.

Description

The **gui_SetSysColor** command sets the color of the specified screen element. The values set in the *redValue*, *greenValue*, and *blueValue* parameters determine the color of the screen element. The values must be between 0 and 255. If all three variables are 0, the color of the screen element will be set to black. If all three variables are 255, the color will be set to white.

Parameters

screenElement – longint. This parameter is a 4D constant that specifies the screen element. See **gui_GetSysColor** for a list of the available constants.

redValue – longint. This parameter is the red component color of the specified screen element.

greenValue – longint. This parameter is the green component color of the specified screen element.

blueValue – longint. This parameter is the blue component color of the specified screen element.

Error Codes

If the function succeeds, the error code return value is one. If the function fails, the error code return value is zero.

gui_SetTrayIcon

gui_SetTrayIcon(action;flags;iconID;processNum;iconHndl;tip; balloon; balloonTitle)→errorCode

COMPATIBILITY: Not available on Windows 95 using 4D 6.7x. Not available using 4D 6.5. If called, returns an errorCode value of –1.

IMPORTANT: Use of this plugin call REQUIRES that IP variable <>ST_TrayNotification be declared.

Variable	Type	Description
<>ST_TrayNotification	longint	This variable will hold the number corresponding to the constants for left mouse button down (TI_LBUTTONDOWN – 513), right mouse button down (TI_RBUTTONDOWN – 516), left mouse button double-click (TI_LBUTTONDBLCLK – 515), and right mouse button double-click (TI_RBUTTONDBLCLK – 518). Test for the value of this variable in an Outside Call event of the target window and take appropriate action. A pop-up menu can be programmed, a plugin call can be sent to display a balloon message, etc.

Parameter	Type	Description
action	longint	[In] Action constant (see below).
flags	longint	Flags that determine what information is displayed.
iconID	longint	[In] Programmer assigned number for the tray icon.
processNum	longint	[In] Process number of process owning the window that receives tray icon messages.
iconHndl	longint	[In] Number assigned to icon image using **gui_GetIcon**.
tip	text	[In] Text of tool tip that displays when mouse hovers over tray icon.
balloon	text	[In] Text that displays in a balloon that can be requested when clicking on the tray icon (only available with Win2K or later and Shell32.dll version 5.0)
balloonTitle	text	[In] Title that appears above balloon text. Title appears in bold (available for Win2K or later and shell32.dll version 5.0 must be available on system).
errorCode	longint	[Out] Error code.

Description

The **gui_SetTrayIcon** places an icon in the system notification area (commonly referred to as the system tray). Subsequent actions can change the icon, tool tip, and balloon text, hide and show the icon, and delete the icon.

Parameters

action – longint. This variable uses the constants defined below to request various actions for the plugin call.

Constant	Description
TI_ADD (0)	Adds a tray icon for a given window. All parameters are required. Tool tip and/or balloon parameters may be empty strings but must be included.
TI_MODIFY (1)	Requests that information about a tray icon be changed. All parameters are required and the iconID MUST have been previously added using the TI_ADD action.
TI_DELETE (2)	Requests that an icon be deleted from the tray. The flags parameter should be set to 0 and the only other required parameters are the iconID.

flags – longint. This variable uses the constants defined below to further define the requested action. The constants should be OR'd together as necessary.

Constant	Description
TI_MESSAGE (1)	Specifies that the tray icon should send a mouse message to the window specified in windowHndl. The plugin responds to right or left mouse button clicks. Double clicks are not supported. Do not use this flag if the tray icon will only be displaying a tool tip and not responding to mouse clicks.
TI_ICON (2)	This flag should be included whenever an icon is to be displayed.
TI_TIP (4)	Include this flag to specify that a tool tip should be displayed when the mouse hovers over the tray icon.
TI_HIDE (256)	Use this flag to hide the tray icon. It is not removed from the tray (although it appears to be removed). Additional flags may also be used if changing icon information.

TI_SHOW (512)	Use this flag to redisplay the tray icon. Additional flags may be used to signal changing icon information.
TI_INFO (16)	Use this flag to display a balloon above the tray icon. There is a 10 second timeout on the balloon.

iconID – longint. Assign a numeric value to this variable that will be used in all subsequent plugin calls that modify or delete this icon. The iconHndl may be changed to show a different icon but the iconID must remain the same.

processNum – longint. Provide the current process number of the process that should receive the outside call. While in most cases this should be the process that controls the window that receives the mouse messages, it doesn't have to be. The process must exist for the outside call to be delivered.

windowHndl – longint. Use the **gui_GetWindow** or **gui_GetWindowFrom4DWin** call to obtain a Windows window handle. By calling **gui_GetWindow** with an empty string, the main 4D window handle is retrieved and thus the icon will be associated with the main window.

iconHndl – longint. Use the **gui_GetIcon** call to obtain a numeric handle to the icon that will be used for the tray icon. This handle can be changed using a modify call.

tip – text. Up to 60 characters may be used for a tool tip that appears when the mouse hovers over the tray icon.

balloon – text. Up to 250 characters may be used for a balloon-style tool tip that can be programmed to appear. The balloon disappears after a fixed interval of 10 seconds. This feature is available on Win2K and later and the workstation must have version 5.0 or later of shell32.dll installed. The plugin tests for both the OS version and the presence of version 5.0 of shell32.

balloonTitle – text. Up to 60 characters may be used for a balloon tip title. The title will appear in bold. If an empty string is provided, then no title will be displayed. If the first character of the text string is a 1, 2, or 3, it will be interpreted to mean the inclusion of an icon to the left of the title. The icons are:

1. *The information icon* – White quotation balloon with a blue "i" inside.
2. *The warning icon* – Yellow triangle with an exclamation point inside.
3. *The error icon* – Red circle with an X. inside.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero or –1 if called on incompatible OS or 4D version (refer to compatibility note above).

Examples

```
`In some startup method
C_LONGINT(<>ST_TrayNotification)

  ` Method using a tray icon
C_LONGINT($lErr;$lAction;$lFlags;$lIconID;lWindow;$lIconHndl)
C_TEXT($tTip;$tBalloonInfo;$tBalloonTitle)

lWindow:=gui_GetWindow ("A Window Title")
$lIconID:=200
$lErr:=gui_LoadIcon ("c:\Temp\myIcon.ico";$lIconHndl)
$tTip:="A Tool Tip"
$tBalloonInfo:="For Win2K and Shell32 v5"
$tBalloonTitle:="2A Warning Msg"
$lAction:=TI_ADD
$lFlags:=TI_MESSAGE  | TI_ICON  | TI_TIP
$lErr:=gui_SetTrayIcon ($lAction;$lFlags;$lIconID;
     Current process;$lIconHndl;$tTip;
     $tBalloonInfo;$tBalloonTitle)
```

Example Form Events

```
C_LONGINT($lErr;$lAction;$lFlags;$lIconID;$lIconHndl)
C_TEXT($tTip;$tBalloonInfo;$tBalloonTitle)

Case of
  : (Form event=On Load )
    C_LONGINT(lWindow)

  : (Form event=On Outside Call )
    Case of
      : (<>ST_TrayNotification=TI_LBUTTONDOWN )
        $lErr:=gui_LoadIcon ("c:\Temp\TrafficRD.ico";$IconHndl)
        ` Hide icon for 5 seconds
        $lErr:=gui_SetTrayIcon (TI_MODIFY;TI_MESSAGE | TI_HIDE
              | TI_ICON ;100;Currentprocess;lWindow;$IconHndl;
              "Red light tooltip";"Balloon Info";"Title")
        DELAY PROCESS(Current process;60*5)
          `Redisplay icon
        $lErr:=gui_SetTrayIcon (TI_MODIFY;TI_ICON | TI_MESSAGE
              | TI_SHOW ;100;Current process;lWindow;$IconHndl;
              "Red light tooltip";"Balloon Info";"Title")
        <>ST_TrayNotification:=0  `Reset for next message

      : (<>ST_TrayNotification=TI_RBUTTONDOWN )
        ARRAY TEXT($atMenuSelections;6)
        C_TEXT($tMenuText)
        C_LONGINT(lPopupSelection)
        $atMenuSelections{1}:="Miles Davis"
        $atMenuSelections{2}:="Lee Ritenour"
        $atMenuSelections{3}:="(-"
```

```
            $atMenuSelections{4}:="Dave Grusin"
            $atMenuSelections{5}:="Bill Evans"
            $atMenuSelections{6}:="New York Voices"
            For ($i;1;Size of array($atMenuSelections))
              $tMenuText:=$tMenuText+";"+$atMenuSelections{$i}
            End for
            ` get rid of the first semicolon
            $tMenuText:=Substring($tMenuText;2)

            lPopupSelection:=Pop up menu($tMenuText)
            <>ST_TrayNotification:=0

          : (Form event=On Unload)
            $lErr:=gui_SetTrayIcon (TI_DELETE;0;100)

        End case
    End case
```

gui_SetWindowLong

gui_SetWindowLong(windowHandle;style;mode;level)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
style	longint	[In] The style to set.
mode	longint	[In] Set mode.
level	longint	[In] What style to set.
errorCode	longint	[Out] Error code.

Description

The **gui_SetWindowLong** function changes an attribute of the specified window. This is almost a direct mapping on the Win32 **SetWindowLong**.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

style – longint. Specify one of the following values:

Normal Styles:

Constant	Description
WS_VISIBLE (268435456)	Makes a window visible.
WS_CAPTION (12582912)	Sets window to have a title bar.
WS_BORDER (8388608)	Sets window to have a border frame.
WS_DLGFRAME (4191304)	Sets window to have a non-sizable dialog frame.
WS_SYSMENU (524288)	Adds or removes the system menu and control buttons from the title bar.
WS_THICKFRAME (262144)	Sets a window to have a thick, sizable frame. This is the default setting.
WS_MINIMIZEBOX (131072)	Adds/removes the minimize button.
WS_MAXIMIZEBOX (65536)	Adds/removes the maximize button.

mode – longint. Specify one of the following values:

Constant	Description
WIN_ENABLE (1)	Enables the selected style.
WIN_DISABLE (0)	Disables the selected style.

level – longint. Specify one of the following values:

Constant	Description
WIN_EXSTYLE (1)	Sets a new extended window style.
WIN_STYLE (0)	Sets a new window style.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Remarks

You may use "OR" (|) to combine some of the styles. If both the minimize box and maximize box are disabled, then the buttons are removed. If only one is disabled, then the corresponding button is colored gray to indicate its inactive status. These behaviors are defined by Windows – the plugin is merely the facilitator. There is no way to remove just one of the controls, as Windows does not allow this.

For all the attributes that are defined here, you should use WIN_STYLE. WIN_EXSTYLE is available, but none of the attributes defined as constants in this plugin are used with WIN_EXSTYLE.

Example

This example will remove both the minimize and restore buttons in the main 4D application window.

```
C_LONGINT($lErr;$lWindowHandle)
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_SetWindowLong ($lWindowHandle;WS_MINIMIZEBOX
        | WS_MAXIMIZEBOX ;WIN_DISABLE ;WIN_STYLE )
End if
```

gui_SetWindowStyle

gui_SetWindowStyle(windowHandle;action)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
action	longint	[In] State to use for displaying the window.
errorCode	longint	[Out] Error code.

Description

The **gui_SetWindowStyle** function sets the specified window's capabilities. It allows the disabling or enabling of the minimize, maximize, close, and resize operations.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

action – longint. This specifies whether to enable or disable particular operations for the window.

Constant	Description
RW_DISABLE_MIN (64)	Disables the minimize button on the specified window.
RW_ENABLE_MIN (128)	Enables the minimize button on the specified window.
RW_DISABLE_MAX (256)	Disables the maximize button on the specified window.
RW_ENABLE_MAX (512)	Enables the maximize button on the specified window.
RW_DISABLE_CLOSE (1024)	Disables the close button on the specified window.
RW_ENABLE_CLOSE (2048)	Enables the close button on the specified window.
RW_DISABLE_RESIZE (4096)	Disables the resize button on the specified window.
RW_ENABLE_RESIZE (8192)	Enables the resize button on the specified window.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Example

This example will disable the minimize button on the 4D application window.

```
C_LONGINT($lErr;$lWindowHandle)
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_SetWindowStyle ($lWindowHandle;RW_DISABLE_MIN )
End if
```

gui_SetWindowTitle

gui_SetWindowTitle(windowHandle;windowTitle)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
windowTitle	string	[In] New window title name.
errorCode	longint	[Out] Error code.

Description

The **gui_SetWindowTitle** call changes the text of the title bar for the specified window.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

windowTitle – string. This is the string to use as the new window title. This parameter may be up to 256 characters long.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Remarks

The **gui_SetWindowTitle** function does not expand tab characters (ASCII code 0x09). Tab characters are displayed as vertical bar (|) characters.

Example

This example will change the name of the main 4th Dimension, 4D Client, or 4D Server application window, as well as the name that appears in the Windows Task Bar, depending on the runtime environment. This code snippet could be called in an application's On Startup method as well as in its On Server Startup method.

```
C_LONGINT($lErr;$lWindowHandle)
C_STRING(80;$s804DAppName)
Case of
  : (Application type=4D Server )
    $s804DAppName:="CoolApp Server"
  : (Application type=4D Client )
    $s804DAppName:="CoolApp Client"
  Else
    $s804DAppName:="CoolApp"
End case
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_SetWindowTitle ($lWindowHandle;$s804DAppName)
End if
```

gui_SetWndRect

gui_SetWndRect(windowHandle;x;y;w;h)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
x	longint	[In] X location.
y	longint	[In] Y location.
w	longint	[In] Window width.
h	longint	[In] Window length.
respectTaskbar	longint	[In] Optional.
errorCode	longint	[Out] Error code.

Description

The **gui_SetWndRect** function changes the size of the window

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

x – longint. Specifies the x-coordinate of the window. This is the distance from the left side of the screen to the outer edge of the window.

y – longint. Specifies the y-coordinate of the window. This is the distance from the top of the screen to the outer edge of the window.

w – longint. Specifies the width of the window. This is the distance from outer left side to outer right side.

h – longint. Specifies the height of the window. This is the distance from the outer top to the outer bottom.

respectTaskbar – longint. Specifies if the new size and location of the window should be allowed to hide the Windows taskbar. Default is 0 (allow hiding of taskbar). Any value >0 will prevent the taskbar from being hidden.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Remarks

This call will only change the size of the window and will not modify its Z-Order (i.e., it will not bring the window to the front).

The current state of the window will not be modified. If it is minimized, then it will remain minimized.

Example

This example relocates the position of the entire 4D application window on the screen by moving it 10 pixels to the right and 10 pixels down, and then shrinks it by 10 pixels in each direction.

```
C_LONGINT($lErr;$lWindowHandle;$x;$y;$w;$h)
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_GetWndRect ($lWindowHandle;$x;$y;$w;$h)
  If ($lErr#0)
    $lErr:=gui_SetWndRect ($lWindowHandle;$x+10;$y+10;$w-10;$h-10)
  End if
End if
```

gui_ShowTaskBar

gui_ShowTaskBar→returnCode

Parameter	Type	Description
returnCode	longint	[Out] Return code.

Description

The **gui_ShowTaskBar** function will show the Windows task bar if it was previously hidden by a call to gui_HideTaskBar.

Error Codes

If the task bar was successfully shown the return code will be a nonzero value. If there was an error showing the task bar the return code will be zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_HideTaskBar

  ` The task bar will now remain hidden
  ` until the following call to gui_ShowTaskBar

$lErr:=gui_ShowTaskBar
```

gui_ShowTitleBar

gui_ShowTitleBar→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_ShowTitleBar** function shows the Windows title bar of the calling application previously hidden with gui_HideTitleBar.

Error Codes

If the title bar was successfully displayed the error code returned is non zero. If the function fails the error code returned is zero.

Example

```
C_LONGINT($lErr)

$lErr:=gui_HideTitleBar

  ` The title bar will now remain hidden
  ` until the following call to gui_ShowTitleBar

$lErr:=gui_ShowTitleBar
```

gui_ShowWindow

gui_ShowWindow(windowHandle;showState)→errorCode

Parameter	Type	Description
windowHandle	longint	[In] Window handle to use.
showState	longint	[In] State to use for displaying the window.
errorCode	longint	[Out] Error code.

Description

The **gui_ShowWindow** function sets the specified window's show state.

Parameters

windowHandle – longint. This is the handle for the window. Use **gui_GetWindow** or **gui_GetWindowFrom4DWin** (see the sections on these commands).

showState – longint. This specifies how the window is to be shown. This parameter is a predefined constant, and can be one of the following values:

Constant	Description
SW_HIDE (0)	Hides the window and activates another window.
SW_MAXIMIZE (3)	Maximizes the specified window.
SW_MINIMIZE (6)	Minimizes the specified window and activates the next top-level window in the Z order.
SW_RESTORE (9)	Activates and displays the window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when restoring a minimized window.
SW_SHOWMAXIMIZED (3)	Activates the window and displays it as a maximized window.
SW_SHOWNA (8)	Displays the window in its current state. The active window remains active.
SW_SHOWNOACTIVATE (4)	Displays a window in its most recent size and position. The active window remains active.

Error Codes

If the function succeeds, the error code return value is non-zero. If the function fails, the error code return value is zero.

Example

This example will procedurally minimize the 4D application window, and then maximize it.

```
C_LONGINT($lErr;$lWindowHandle)
$lWindowHandle:=gui_GetWindow ("")  ` main 4D window
If ($lWindowHandle>0)
  $lErr:=gui_ShowWindow ($lWindowHandle;SW_MINIMIZE )
  $lErr:=gui_ShowWindow ($lWindowHandle;SW_MAXIMIZE )
End if
```

gui_SubClassInit

gui_SubClassInit(action)→errorCode

Parameter	Type	Description
action	longint	[In] Action to perform.
errorCode	longint	[Out] Error code.

Description

The **gui_ SubClassInit** function works only with **gui_RestrictWindow** to subclass all child windows and allow the interception of commands that pertain to changing the window behavior. See the **gui_RestrictWindow** topic for more information on using this command.

Parameters

action – longint. This is the action to perform. This parameter is a predefined constant, and can be one of the following values:

Constant	Description
RW_SUBCLASS_INIT (1024)	Initializes the subclass.
RW_RELEASE (0)	Releases the subclass information.

Remarks

Use the **gui_ SubClassInit** function only with **gui_RestrictWindow**

Error Codes

The function returns zero if the function fails or non-zero value if the function succeeds.

Example

```
C_LONGINT($lErr;$lWindowHandle;$lRestrictions)
$lErr:=gui_SubClassInit (RW_SUBCLASS_INIT )
If ($lErr#0)
  $lRestrictions:=RW_NO_SIZE  ^| RW_NO_MOVE
  $lWindowHandle:=gui_GetWindow ("MyWindow")
  $lErr:=gui_RestrictWindow ($lWindowHandle;$lRestrictions)
  `window can't be moved or sized
End if
```

gui_TakeScreenShot

gui_TakeScreenShot(windowHandle;fileName)→errorCode

Parameter	Type	Description
windowHandle	longint	[IN] Handle to the window to capture.
fileName	text	[IN] Full file path of BMP to save to.
errorCode	longint	[Out] Error code.

Description

The **gui_TakeScreenshot** command will take a screenshot of the window passed in and save it as a bitmap image at the provided file path.

Parameters

windowHandle – longint. The handle to the window to capture. See **gui_GetWindow()**.

fileName – text. The full file path and name of the bitmap where the screenshot should be saved. For example, "C:\\Users\\me\\desktop\\picture.bmp".

Error Codes

If the function succeeds, it will return zero. If not, it will return one of the non-zero values below:

1 = Failed to create a compatible device context.

2 = Failed to create a compatible bitmap.

3 = Failed to transfer the bit blocks to the device context in memory.

4 = Invalid window handle.

Example

```
C_LONGINT($lWindowHandle)
C_BLOB($xBlob)
C_PICTURE(gScreenshot)
C_TEXT($tFilePath)

$tFilePath:=Temporary folder+"screenshot.bmp"
$lWindowHandle:=gui_GetWindow (Get window title)
$lErr:=gui_TakeScreenshot ($lWindowHandle;$tFilePath)

If($lErr=0)
      DOCUMENT TO BLOB($tFilePath;$xBlob)
      BLOB TO PICTURE($xBlob;gScreenshot)
      DELETE DOCUMENT($tFilePath)
Else
      util_alert("Failed to take screenshot")
End if
```

gui_ToolTip Methods

gui_ToolTipCreate(style;handle)→errorCode

COMPATIBILITY: The tool tip methods require Comctl32.dll version 4.70 or later. These functions are ONLY available for the Windows 95 and Windows NT operating systems with this (and higher) version DLL. The DLL is installed with IE version 5.0 and higher or it may be installed manually. A test is made within the call for the appropriate version. If not correct, the call immediately exits with an error code of zero.

Parameter	Type	Description
style	longint	[In] Balloon style or rectangular.
handle	longint	[In] Optional. Supply handle when the target window does not have current focus. The window handle obtained using **gui_GetWindow**. [Out] Use this window handle as the id in calls to **gui_ToolTipShowOnObject** and **gui_ToolTipShowOnCoord**.
errorCode	longint	[Out] Error code.

Description

The **gui_ToolTipCreate** function establishes a control container to which all tool tips belong. It must be called before any other tool tip functions.

Parameters

style – longint. Constant indicating whether a balloon style or rectangular tool tip should be created.

Constant	Description
TT_BALLOON (0)	The control created will always use tool tips with a balloon style. Coordinates supplied in subsequent calls determine where the balloon tip points.
TT_RECTANGLE (1)	The control created will always use a rectangular style message area.

handle – longint. Optional. Window handle returned by **gui_GetWindow** or **gui_GetWindowFrom4D**.

Error Codes

The function returns zero on failure and a non-zero value upon success.

gui_ToolTipDestroyControl→errorCode

Parameter	Type	Description
errorCode	longint	[Out] Error code.

Description

The **gui_ToolTipDestroyControl** function releases the resources held by the tool tip control.

Parameters

This method has no parameters.

Error Codes

The function returns zero on failure and a non-zero value upon success.

Remarks

No parameters are required. Use this function when no further display of tool tips will be required in the application session.

gui_ToolTipHide(id)→errorCode

Parameter	Type	Description
id	longint	[In] Application defined (programmer supplied) ID. IDs must be in the range 1-500. Any ID number greater than 500 is assumed to be the same window handle used as the second parameter in **gui_ToolTipCreate**. This allows a tool tip to display on a window different than the one that has current application focus.
errorCode	longint	[Out] Error code.

Description

The **gui_ToolTipHide** function hides a tool tip but does not remove it from the control. The tip may again be displayed. The message and location may be changed using **gui_ToolTipShowOnCoord** or **gui_ToolTipShowOnObject**.

Parameters

id – longint. Programmer supplied ID between 1 and 500.

Error Codes

The function returns zero on failure and a non-zero value upon success.

gui_ToolTipShowOnCoord(id;messageText;x-coord;y-coord;howToClose; titleText; methodText;messageBoxWidth)→errorCode

Parameter	Type	Description
id	longint	[In] Application defined (programmer supplied) ID. IDs must be in the range 1-500. Any ID number greater than 500 is assumed to be the same window handle used as the second parameter in **gui_ToolTipCreate**. This allows a tool tip to display on a window different than the one that has current application focus.
messageText	text	[In] Text for tool tip. Also used for title text.
x-coord	longint	[In] Pixel location relative to 4D's main window client area.
y-coord	longint	[In] Pixel location relative to 4D's main window client area.
howToClose	longint	[In] Use the constant TT_CLOSE_ON_CLICK (7) to set the tool tip to be dismissed when either the tool tip or process window is clicked. Any other value will cause the tool tip to remain on screen until a **gui_ToolTipHide** call is made.
titleText	*text*	[In] Text to be used as a message box title. You may display an icon to the left of the title by using a 1, 2, or 3 as the first character of the titleText. The icons available are: 1. Information icon, 2. Warning icon, and 3. Error icon.
methodText	text	[In] Text of a method to be called when the tool tip is clicked. This is available only when the howToClose parameter is TT_CLOSE_ON_CLICK.
messageBoxWidth	longint	[In] Width in pixels for the message box. This is required when messageText is over 80 characters or if the message box should word wrap the message Text.
errorCode	longint	[Out] Error code.

Description

The **gui_ToolTipShowOnCoord** displays a message in a rectangular or balloon style popup near the absolute coordinates provided.

Parameters

id – longint. This is a programmer supplied ID between 1 and 500.

messageText – text. This is the text used for the tool tip as well as the title text.

x-coord – longint. This is a horizontal pixel location relative to 4D's main window client area.

y-coord – longint. This is a vertical pixel location relative to 4D's main window client area.

howToClose – longint. Constant determining how the tool tip can be closed.

TT_CLOSE_ON_CLICK (7)	Sets the tool tip to be dismissed when either the tool tip or process window is clicked. Any other value will cause the tool tip to remain on screen until a **gui_ToolTipHide** call is made.

titleText – text. This is the message box title text. You may display an icon to the left of the title by using a 1, 2, or 3 as the first character of the titleText. The icons available are:

1. *The information icon* – White quotation balloon with a blue "i" inside.
2. *The warning icon* – Yellow triangle with an exclamation point inside.
3. *The error icon* – Red circle with an X. inside.

methodText – text. This method is called when the tool tip is clicked. This is available only when the howToClose parameter is TT_CLOSE_ON_CLICK.

messageBoxWidth – longint. This is the pixel width of the message box. This is required when messageText is over 80 characters or if the message box should word wrap the messageText.

Error Codes

The function returns zero on failure and a non-zero upon success.

gui_ToolTipShowOnObject(id;messageText;location;howToClose;titleText;methodText;left;top;right;bottom;messageBoxWidth)→errorCode

Parameter	Type	Description
id	longint	[In] Application defined (programmer supplied) ID. IDs must be in the range 1-500. Any ID number greater than 500 is assumed to be the same window handle used as the second parameter in **gui_ToolTipCreate**. This allows a tool tip to display on a window different than the one that has current application focus.
messageText	text	[In] Text for tool tip. Also used for title text.
location	longint	[In] Constant indicating location of tool tip.

howToClose	longint	[In] Use the constant TT_CLOSE_ON_CLICK (7) to set the tool tip to be dismissed when either the tool tip or process window is clicked. Any other value will cause the tool tip to remain on screen until a **gui_ToolTipHide** call is made.
titleText	text	[In] Text to be used as a message box title. You may display an icon to the left of the title by using a 1, 2, or 3 as the first character of the titleText. The icons available are: 1. Information icon, 2. Warning icon, and 3. Error icon.
methodText	text	[In] Text name of a 4D method to be called when the tool tip is clicked. This is available only when the howToClose parameter is TT_CLOSE_ON_CLICK.
left	longint	[In] Position of object's left border as returned by the 4D command Get Object Rect.
top	longint	[In] Position of object's top border as returned by the 4D command Get Object Rect.
right	longint	[In] Position of object's right border as returned by the 4D command Get Object Rect.
bottom	longint	[In] Position of object's bottom border as returned by the 4D command Get Object Rect.
messageBoxWidth	longint	[In] [Optional] Width in pixels for the message box. This is required when messageText is over 80 characters or if the message box should word wrap the message Text.
errorCode	longint	[Out] Error code.

Description

The **gui_ToolTipShowOnObject** displays a message in a rectangular or balloon style popup on the target form object.

Parameters

id – longint. This is a programmer supplied ID between 1 and 500.

messageText – text. This is the text used for the tool tip as well as the title text.

location – longint. This is a constant indicating where the tool tip should be placed on the object.

Constant	Description
TT_CENTER (2)	This is the default location. The tool tip will point to the center of the form object whose coordinates were passed to the plugin call. If coordinates are not passed into the plugin call, there must be four process variables defined and used in the 4D command **Get Object Rect** prior to calling this plugin command. For example: **Get Object Rect** (variableName;TT_Left; TT_Top;TT_Right;TT_Bottom)
TT_TOPRIGHT (3)	The tool tip will point to the top, right corner of the object.
TT_TOPLEFT (4)	The tool tip will point to the top, left corner of the object.
TT_BOTTOMRIGHT (5)	The tool tip will point to the bottom, right corner of the object.
TT_BOTTOMLEFT (6)	The tool tip will point to the bottom, left corner of the object.

howToClose – longint. Constant determining how the tool tip is closed.

Constant	Description
TT_CLOSE_ON_CLICK (7)	Sets the tool tip to be dismissed when either the tool tip or process window is clicked. Any other value will cause the tool tip to remain on screen until a **gui_ToolTipHide** call is made.

titleText – text. This is the message box title text. You may display an icon to the left of the title by using a 1, 2, or 3 as the first character of the titleText. The icons available are:

1. *The information icon* – White quotation balloon with a blue "i" inside.
2. *The warning icon* – Yellow triangle with an exclamation point inside.
3. *The error icon* – Red circle with an X. inside.

methodText – text. This method is called when the tool tip is clicked. This is available only when the howToClose parameter is TT_CLOSE_ON_CLICK.

left – longint. This is the position of the object's left border as returned by the 4D command Get Object Rect.

top – longint. This is the position of object's top border as returned by the 4D command Get Object Rect.

right – longint. This is position of object's right border as returned by the 4D command Get Object Rect.

bottom – longint. This is position of object's bottom border as returned by the 4D command Get Object Rect.

messageBoxWidth – longint. This is the pixel width of the message box. This is required when messageText is over 80 characters or if the message box should word wrap the message Text.

Error Codes

The function returns zero on failure and a non-zero value upon success.

Example

```
C_LONGINT($id;$lErr)
C_TEXT($tTip;$tText)

$id:=1
$tTip:="Don't speak unless you can improve the silence."
$lErr:=gui_ToolTipCreate (TT_BALLOON )  `balloon

GET OBJECT RECT(bCheckYears;TT_Left;TT_Top;TT_Right;TT_Bottom)

$tText:="2Words of Wisdom"
$lErr:=gui_ToolTipShowOnObject ($id;$tTip;TT_CENTER
      ;TT_CLOSE_ON_CLICK ;$tText;"methodToExecute")

  `when finished with tool tip
$lErr:=gui_ToolTipHide ($id)

  `when no more tool tips will be displayed
$lErr:=gui_ToolTipDestroyControl ()
```

gui_WinHelp

The **gui_WinHelp** call is not currently implemented. Use the equivalent 4D and ACI Pack commands.

TWAIN Methods

TWAIN_AcquireImage

TWAIN_AcquireImage(allowDialog;xYourBlob)→errorCode

Parameter	Type	Description
allowDialog	longint	[In] Specify to show the device specific TWAIN dialog.
xYourBlob	BLOB	[Out] Optional BLOB parameter to use instead of the required xTWAINBLOB. Must be a process variable and the size must be set to a size greater than zero before call.
errorCode	longint	[Out] Error code.

Description

The **TWAIN_AcquireImage** call retrieves an image from the current TWAIN device opened with TWAIN_SetSource. The retrieved image is placed in the xTWAINBLOB process variable, which must be declared in 4D prior to calling this method.

Parameters

allowDialog – longint. Pass a value greater than zero to display the device specific TWAIN dialog for importing an image. Passing a value less than 1 will suppress the dialog and the TWAIN device will return the default image (usually the most recent).

xYOURBlob – BLOB. Pass a process level blob with a size that is greater than zero to use instead of xTWAINBLOB. This BLOB will then receive the retrieved image.

Error Codes

If the function succeeds the error code return value is one. If *allowDialog* is greater than zero and the user cancels from the dialog, the error code return value is zero. If the function fails the error code return value is less than zero.

Example

```
C_LONGINT($lErr)
C_BLOB(xTWAINBLOB)

$lErr:=TWAIN_AcquireImage (1)  ` Display the TWAIN dialog

Case of
      : ($lErr=1)
            ALERT("Image acquired.  (Size = "+String(BLOB
                        size(xTWAINBLOB))+").")
      : ($lErr=0)
```

```
            ALERT("Image acquisition cancelled by user.")
      Else
            ALERT("Failed to acquire image.")
End case
```

Example #2

```
C_LONGINT($lERR)
C_BLOB(xYOURBLOB)

SET BLOB SIZE(xYOURBLOB;1)

$lERR:=TWAIN_AcquireImage(1;xYOURBLOB) // Display the TWAIN
dialog and also use the xYOURBLOB parameter

Case of
      : ($lErr=1)
            ALERT("Image acquired.  (Size = "+String(BLOB
                        size(xYOURBLOB))+").")
      : ($lErr=0)
            ALERT("Image acquisition cancelled by user.")
      Else
            ALERT("Failed to acquire image.")
End case
```

TWAIN_GetSources

TWAIN_GetSources(sources;debug)→errorCode

Parameter	Type	Description
sources	array text	[Out] Array of TWAIN devices found on the workstation.
dialog	longint	[In] Allow the standard TWAIN dialog.
errorCode	longint	[Out] Error code.

Description

The **TWAIN_GetSources** call retrieves an array of all TWAIN compatible devices available on the workstation.

Parameters

sources – text array. This is the text array variable initialized to zero elements. It is used to receive the list of available TWAIN devices.

dialog – Determines if the standard TWAIN dialog listing devices is displayed. A value of zero will suppress the dialog, any value greater than zero will allow it

Error Codes

When *dialog* is greater than zero , the error code is the ID of the TWAIN device selected by the user or negative one if no device was selected. Otherwise error code is 1 if the function succeeds and 0 if it fails.

Example

```
C_LONGINT($lErr;$i)
C_TEXT($tSourcesFound)

ARRAY TEXT($atSources;0)
$lErr:=TWAIN_GetSources ($atSources)

If ($lErr=1)
      For ($i;1;Size of array($atSources))

      $tSourcesFound:=$tSourcesFound+Char(13)+$atSources{$i}
      End for
      ALERT("The following TWAIN devices were found:
            "+Char(13)+$tSourcesFound)
Else
      ALERT("No TWAIN devices were found.")
End if
```

TWAIN_SetSource

TWAIN_SetSource(source)→errorCode

Parameter	Type	Description
source	text	[In] TWAIN source to enable, as returned from TWAIN_GetSources.
errorCode	longint	[Out] Error code.

Description

The **TWAIN_SetSource** call opens a connection to the specified TWAIN device.

Parameters

source – text. This is the TWAIN source to enable. It is one of the values returned by TWAIN_GetSources.

Error Codes

If the function succeeds the error code return value is one, otherwise it is zero.

Example

```
C_TEXT($1;$tSource)

C_LONGINT($lErr)

$tSource:=$1

C_LONGINT($lErr)

$lErr:=TWAIN_SetSource ($tSource)

If ($lErr=1)
      ALERT($tSource+" successfully enabled.")
Else
      ALERT("Failed to enable"+$tSource)
End if
```

Constants and Values

This section lists the constants used by the Win32API plugin and the values these constants map to. You may find this useful if you use the plugin with versions of 4th Dimension earlier than 6.5.

Constant	Value
BM_CLOSE	“”
BM_SCALE	2
BM_SCALETOMAXCLIENT	3
BM_TILE	1
CL_DRAGDROP	1
COLOR_SCROLLBAR	0
COLOR_DESKTOP	1
COLOR_ACTIVECAPTION	2
COLOR_INACTIVECAPTION	3
COLOR_MENU	4
COLOR_WINDOW	5
COLOR_WINDOWFRAME	6
COLOR_MENUTEXT	7
COLOR_WINDOWTEXT	8
COLOR_CAPTIONTEXT	9
COLOR_ACTIVEBORDER	10
COLOR_INACTIVEBORDER	11
COLOR_APPWORKSPACE	12
COLOR_HIGHLIGHT	13
COLOR_HIGHLIGHTTEXT	14
COLOR_3DFACE	15
COLOR_3DSHADOW	16
COLOR_GRAYTEXT	17
COLOR_BTNTEXT	18
COLOR_INACTIVECAPTIONTEXT	19
COLOR_3DHIGHLIGHT	20
COLOR_3DDKSHADOW	21
COLOR_3DLIGHT	22
COLOR_INFOTEXT	23
COLOR_INFOBK	24
COLOR_HOTLIGHT	26
COLOR_GRADIENTACTIVECAPTION	27
COLOR_GRADIENTINACTIVECAPTION	28
DT_FORCE_UPDATE	1
EP_NAMES_ONLY	1
EP_USE_OPEN	2
EP_USE_REGISTRY	0
ERROR_ACCESS_DENIED	5
ERROR_FILE_EXISTS	80

Constant	Value
ERROR_FILE_NOT_FOUND	2
ERROR_PATH_NOT_FOUND	3
ERROR_SHARING_VIOLATION	32
FD_CREATE_PROMPT	8192
FD_DISABLE_EDIT_FIELD	16384
FD_DISABLE_LOOKIN_FIELD	256
FD_FILE_MUST_EXIST	4096
FD_FILES_ONLY	4
FD_HIDE_NEWDIRECTORY_BUTTON	1024
FD_HIDE_TOOLBAR	512
FD_HIDE_UP_BUTTON	32768
FD_OVERWRITE_PROMPT	2
FD_SELECT_DIRECTORY	2048
FLASHW_ALL	3
FLASHW_BRING_TO_FOREGROUND	12
FLASHW_CAPTION	1
FLASHW_STOP	0
FLASHW_TIMER	4
FLASHW_TRAY	2
GR_HKEY_CLASSES_ROOT	1
GR_HKEY_CURRENT_USER	2
GR_HKEY_DYN_DATA	3
GR_HKEY_LOCAL_MACHINE	4
GR_HKEY_USERS	5
GR_HKEY_CURRENT_CONFIG	6
GR_HKEY_PERFORMANCE_DATA	7
GR_TYPE_BINARY	1
GR_TYPE_LONGINT	2
GR_TYPE_TEXT	3
GR_TYPE_ARRAYTEXT	4
HELP_CONTEXT	1
HELP_CONTEXTPOPUP	8
HELP_FINDER	11
HELP_HELPONHELP	4
HELP_INDEX	3
HELP_QUIT	2
HELP_SETINDEX	5
HELP_TCARD	32768
IS_MINIMIZED	1
IS_MAXIMIZED	2
LANG_DUTCH	19
LANG_ENGLISH	9
MB_DEFBUTTON2	256
MB_DEFBUTTON3	512
MB_DEFBUTTON4	756

Constant	Value
MB_ABORTRETRYIGNORE	2
MB_APPLMODAL	0
MB_CANCELTRYCONTINUE	6
MB_DEFBUTTON1	0
MB_ICONINFORMATION	64
MB_ICONQUESTION	32
MB_ICONSTOP	16
MB_ICONWARNING	48
MB_IDABORT	3
MB_IDCANCEL	2
MB_IDCONTINUE	11
MB_IDIGNORE	5
MB_IDNO	7
MB_IDOK	1
MB_IDRETRY	4
MB_IDTRYAGAIN	10
MB_IDYES	6
MB_OKCANCEL	1
MB_RETRYCANCEL	5
MB_SYSTEMMODAL	4096
MB_TASKMODAL	8192
MB_YESNO	4
MB_YESNOCANCEL	3
MB_ASTERISK	64
MB_EXCLAMATION	48
MB_OK	0
MB_QUESTION	32
OS_ME	3
OS_NT351	351
OS_NT4	400
OS_W2K	500
OS_WIN95	1
OS_WIN98	2
OS_WIN03	520
OS_XP	510
OS_VISTA_LONGHORN	600
OS_SERVER2K8	601
OS_WIN7	610
OS_SERVER2K8R2	611
OS_WIN8	620
OS_SERVER2012	621
PS_COPIES	4
PS_PORTRAITORLANDSCAPE	5
PS_PRINTEDTOFILE	6
PS_PRINTER	1

Constant	Value
PS_PRINTPREVIEW	7
PS_SIZE	2
PS_SOURCE	3
RS_AMSYMBOL	6
RS_CURRENCYDECIMALSYMBOL	15
RS_CURRENCYDIGITSAFTERDECIMAL	16
RS_CURRENCYGROUPINGSYMBOL	17
RS_CURRENCYSYMBOL	14
RS_DATESEPARATOR	3
RS_DECIMALSYMBOL	9
RS_DIGITSAFTERDECIMAL	11
RS_LISTSEPARATOR	18
RS_LONGDATEFORMAT	2
RS_MEASURESYSTEM	8
RS_NEGATIVESYMBOL	13
RS_NUMBERGROUPINGSYMBOL	12
RS_NUMBERLEADINGZEROS	10
RS_PMSYMBOL	7
RS_SHORTDATEFORMAT	1
RS_TIMEFORMAT	4
RS_TIMESEPARATOR	5
RW_DISABLE_CLOSE	1024
RW_DISABLE_MAX	256
RW_DISABLE_MIN	64
RW_DISABLE_RESIZE	4096
RW_ENABLE_CLOSE	2048
RW_ENABLE_MAX	512
RW_ENABLE_MIN	128
RW_ENABLE_RESIZE	8192
RW_NO_MAX	8
RW_NO_MIN	4
RW_NO_MOVE	2
RW_NO_SIZE	1
RW_RELEASE	0
RW_SUBCLASS_INIT	1024
SC_HAS_CUSTOM_COLORS	1
SC_NO_CUSTOM_COLORS	0
SW_HIDE	0
SW_MAXIMIZE	3
SW_MINIMIZE	6
SW_RESTORE	9
SW_SHOW	5
SW_SHOWMAXIMIZED	3
SW_SHOWMINNOACTIVE	7
SW_SHOWMINIMIZED	2

Constant	Value
SW_SHOWNA	8
SW_SHOWNOACTIVATE	4
SW_SHOWNORMAL	1
TI_ADD	0
TI_DELETE	2
TI_HIDE	256
TI_ICON	2
TI_INFO	16
TI_LBUTTONDBLCLK	515
TI_LBUTTONDOWN	513
TI_MESSAGE	1
TI_MODIFY	1
TI_RBUTTONDBLCLK	518
TI_RBUTTONDOWN	516
TI_SHOW	512
TI_TIP	4
TT_BALLOON	0
TT_BOTTOM	Object Get Rectangle
TT_BOTTOMLEFT	6
TT_BOTTOMRIGHT	5
TT_CENTER	2
TT_CLOSE_ON_CLICKED	7
TT_LEFT	Object Get Rectangle
TT_RECTANGLE	1
TT_RIGHT	Object Get Rectangle
TT_TOP	Object Get Rectangle
TT_TOPLEFT	4
TT_TOPRIGHT	3
WIN_DISABLE	0
WIN_ENABLE	1
WIN_EXSTYLE	1
WIN_FALSE	0
WIN_STYLE	0
WIN_TRUE	1
WM_BORDER_HEIGHT	6
WM_BORDER_WIDTH	5
WM_CAPTION_HEIGHT	4
WM_MENU_HEIGHT	15
WS_BORDER	8388608
WS_CAPTION	12582912
WS_CHILD	1073741824
WS_CLIPCHILDREN	33554432
WS_CLIPSIBLINGS	67108864
WS_DISABLED	134217728
WS_DLGFRAME	4191304

Constant	Value
WS_HSCROLL	1048576
WS_MAXIMIZEBOX	65536
WS_MINIMIZEBOX	131072
WS_SYSMENU	524288
WS_THICKFRAME	262144
WS_VISIBLE	268435456
WS_VSCROLL	2097152