yesFALSEyesCTRLMGRControl Manageryesyesyesyesyes

Table of Contents

cSetTVItem
cSetUpDownPos
cGetCBCount
CSetWndText
cWinIdConvert
cWndByClass
cWndByID
cWndByName
cWndBySeq
cWndByWndSpec
cWndExist
cWndInfo
cWndGetWndSpec
cWndState

Control Manger Analysis Tutorial
What is the Control Manager Extender?
Overview of Control Manager Usage
Types Of Controls
Tutorial One - Entering text into an EditBox Control
Tutorial Two

Partial Window Names
RoboScripter
Options for designing your script
RoboScripter Tutorial

CONTROL MANAGER EXTENDER HELP

Function lists

Alphabetical: Function list

Lists the functions in alphabetical order.

Control Type: Function List

Lists the functions in order, by the control type.

Options for designing your script:

RoboScripter

RoboScripter Tutorial

This 'RoboScripter Tutorial' is intended to be a step by step guide to aid in the understanding of how the Control Manager Extender, and the RoboScripter, are used to generate code for a WIL script.

Window Analysis script

Control Manger Analysis Tutorial

This 'Control Manger Analysis Tutorial' is intended to be a step by step guide to aid in the understanding of how the Control Manager Extender, and its functions, are incorporated into a WIL script.

Notational Conventions

Contacting Wilson WindowWare

How to get Technical Support

About this Help File

The **Control Manager Extender** allows near-complete access to all standard Windows controls displayed on the screen, and especially within Dialog Boxes presented by various applications.

Controls include Check Boxes, Radio Buttons, Tabbed Dialogs, List Boxes and Combo Boxes.

-Check Boxes can be interrogated, checked, or unchecked.

-Radio buttons can be interrogated or set.

-A particular tab in a Tabbed Dialog can be selected.

-Contents of List and Combo Boxes can be interrogated and specific items within the List or Combo Box can be selected.

Each function has a complete running example to show its usage.

What is the Control Manager Extender?

Overview of Control Manager Usage

Types of Controls

Contacting Wilson WindowWare

Wilson WindowWare, Inc.
5421 California Ave. SW
Seattle, WA 98136 USA

Orders: (800) 762-8383
Voice: (206) 938-1740
Fax: (206) 935-7129

Email: info@windowware.com

Registered users of our software receive manuals, technical support, use of Wilson WindowWare on-line information services, and special offers on new versions of Wilson WindowWare software products.

Alphabetical: Function List

Control Type: Function List

How to Get Technical Support

How to get Technical Support

The Wilson WindowWare web site is an excellent technical resource. Access to the entire Technical Support Database is at your fingertips. In the Technical Support area, use the keyword search to find answers to common problems, alternate scripting methods, and sample code. Or join the Wilson WindowWare Web BBS, a new Web forum. The BBS provides an outlet for registered users to share their experiences with other users.

See the information on registering a copy (found in the WinBatch.hlp file).

The latest versions of our software are available on-line. The places here may change at any time -- check the installation sheet for the most recent addresses.

Internet Web page: http://www.windowware.com

Internet Technical Support Articles & Web BBS:
http://techsupt.windowware.com
http://webboard.windowware.com

Internet FTP: ftp.windowware.com in /wwwftp

Contacting Wilson WindowWare

Alphabetical: Function List

Control Type: Function List

About this Help File

This extender adds certain network capability to the Windows Interface Language (WIL) processing engine. Please refer to the **WIL Reference Manual** for an introduction to WIL, as well as for complete documentation of the many functions available in WIL and the programs that use it. This help file includes only topics and functions which are exclusive to this particular WIL Extender.

Notational Conventions

Throughout this manual, we use the following conventions to distinguish elements of text:

ALL-CAPS
: Used for filenames.

Boldface
: Used for important points, programs, function names, and parts of syntax that must appear as shown.

`system`
: Used for items in menus and dialogs, as they appear to the user.

`Small fixed-width`
: Used for WIL sample code.

Italics
: Used for emphasis, and to liven up the documentation just a bit.

Acknowledgments

This extender developed by Morrie Wilson and Tony Deets.

Documentation written by Deana Dahley, Tony Deets, and Laura Plaut.

Options for designing your script

You have two options for designing your script.

RoboScripter

A Control Manger Extender companion program, with a GUI interface, that is designed to automatically generate code, that interfaces with standard windows controls.

See the RoboScripter Tutorial, for more details.

Windows Analysis script

The Windows Analysis script (analysis.wbt) creates a report (trash.txt) on the windows visible at the time the report is run.

The basic technique is to open the application and get the dialog boxes you are interested in displayed on the screen.

This Window Analysis script will create / output a text file detailing the contents of the window in question.

See the Control Manager Analysis Tutorial, for more details.

RoboScripter

Robo Scripter Tutorial

Contents

RoboScripter is a Control Manger Extender companion program designed to automatically generate code, that interfaces with standard windows controls.

It allows near-complete access to all standard Windows controls displayed on the screen, and especially within Dialog Boxes presented by various applications.

Control types include: Check Boxes, Radio Buttons, Tabbed Dialogs, List Boxes and Combo Boxes.

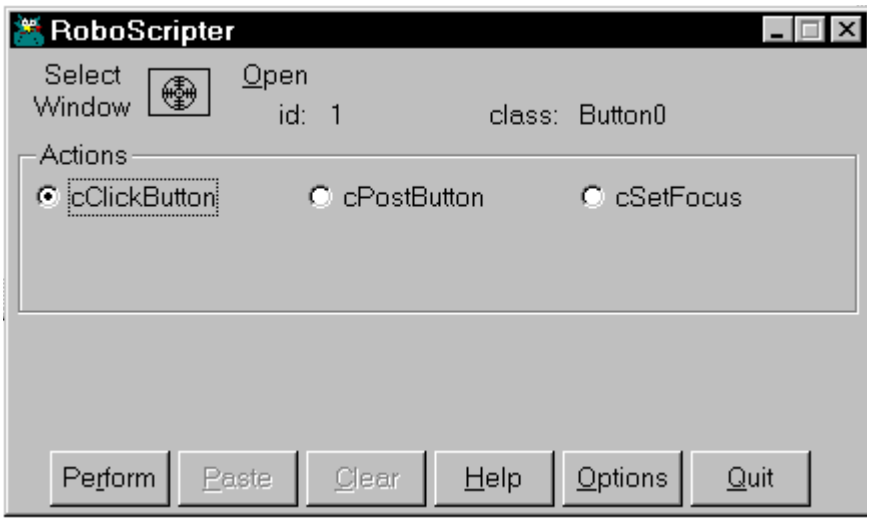
Check Boxes can be interrogated, checked, or unchecked.

Radio buttons can be interrogated or set.

A particular tab in a Tabbed Dialog can be selected.

Contents of List and Combo Boxes can be interrogated and specific items within the List or Combo Box can be selected.

The following is a RoboScripter dialog. Click on the controls in the dialog for pop-up help.



How do I use it?

The basic technique is to run RoboScripter from the Windows 'Start Menu'.
To launch RoboScripter select 'Start | Programs | WinBatch | RoboScripter.

RoboScripter will prompt you to choose the application, you want to automate. RoboScripter will then launch that application. If you choose not to launch the application, select the 'Cancel' button on the 'Run Program' dialog.

You can then drag the cross-hairs to select the target window you want to automate. Choose the appropriate 'Action', then select the 'Perform' button.

Continue this process until you are finished. When finished, select the 'Paste' button, which pastes the generated code into the clipboard. Press the 'Quit' button when complete.

Now, launch the WinBatch Studio editor. Type 'ctrl-v' to paste the generated code into a file. Save the file with the extension '.WBT'. Now you can either edit or run the script.

Windows Analysis script

Control Manager Analysis Tutorial

Alphabetical: Function List

Control Type: Function List

What does the Windows Analysis script do?

The Windows Analysis script (analysis.wbt) creates a report (trash.txt) on the windows visible at the time the report is run.

The basic technique is to open the application and get the dialog boxes you are interested in displayed on the screen.

This Window Analysis script will create / output a text file detailing the contents of the window in question.

How do I use it?

Run the script, by double clicking on analysis.wbt (which is most likely located in the WinBatch\Samples subdirectory). A dialog with all the visible window titles, will appear. Choose the window of interest. The Window Analysis script will process and output a file called 'trash.txt' and load it into Notepad for you to view.

Each window and control has its own unique window handle (often seen as hWnd or Wnd). Use the output of the Window Analysis script and the various cWndBy... functions in this extender to obtain the handle to the window or control you need, then use the specific functions designed for each type of control to either obtain information about the item, or to set it to the desired value.

The relationships, names, Ids, and exactly what windows controls are on the screen can get confusing. To help sort it out, run your application, and get to the point where the windows of interest are visible on the screen. Then run this script. It will create a report detailing the relationships between the windows and controls in your application. You can usually pick off window IDs, titles, and sequence numbers from the list to use in the Control Manager extender functions.

The IDs are usually the best to use if possible since they are unique, then the titles, then the sequence numbers. The sequence numbers are the most likely to change unexpectedly...

```
; This script creates a report on the windows visible at the time
; the report is run.  Run the script.  Choose a top-level
; window of interest, and it will show pertinent information
; that may be necessary to use the Control Manager extender.

; Note that Tabbed dialogs sometimes must be displayed before
; their controls are brought into existence.  So when using
; tabbed dialogs, tab to the correct dialog first.  The
; cWndInfo example shows how to move thru a tabbed dialog.

AddExtender("wwctl34i.dll")
fname="trash.txt"
```

```
a=WinItemize()
b=AskItemList("Choose a Window",a,@tab,@unsorted,@single)
hwnd=DllHwnd(b)
myfile=FileOpen(fname,"WRITE")

workwnd=hwnd
gosub gettextandclass
FileWrite(myfile,"P   C   C   C   C")
FileWrite(myfile,"A   H   H   H   H")
FileWrite(myfile,"R   I   I   I   I")
FileWrite(myfile,"E   L   L   L   L")
FileWrite(myfile,"N   D   D   D   D")
FileWrite(myfile,strcat(strfix("T   2   3   4   5"," ",21),strfix("CLASS","
",26),strfix("IDENT"," ",7),"TITLE"))

FileWrite(myfile,strfill("-",80))
FileWrite(myfile,strcat(strfix("TOP"," ",21),strfix(class," ",26),strfix(ident,"
",7),text))

gosub ProcessChildren
FileClose(myfile)

Run("notepad.exe",fname)
exit

:ProcessChildren
nextchild=cWndInfo(workwnd,8)
ChildCount=0
if nextchild==0
   FileWrite(myfile,"    X               NONE")
else
   while nextchild
       ChildCount=ChildCount+1
       workwnd=nextchild
       gosub gettextandclass
       FileWrite(myfile," ")
       FileWrite(myfile,strcat("    ",strfix(ChildCount," ",17),strfix(class,"
",26),strfix(ident," ",7),text))

       nextchild=cWndInfo(workwnd,6)
       gosub ProcessGrandchildren    ; note this destroys workwnd variable....
   endwhile
endif
return

:ProcessGrandchildren
grandchild=cWndInfo(workwnd,8)
GrandCount=0
if grandchild==0
   FileWrite(myfile,"        X           NONE")
else
   while grandchild
       GrandCount=GrandCount+1
       workwnd=grandchild
       gosub gettextandclass
       FileWrite(myfile,strcat("        ",strfix(GrandCount," ",13),strfix(class,"
",26),strfix(ident," ",7),text))

       grandchild=cWndInfo(workwnd,6)
       gosub ProcessGreatGrandChildren
   endwhile
endif
```

```
return

:ProcessGreatGrandchildren
greatgrandchild=cWndInfo(workwnd,8)
greatGrandCount=0
if greatgrandchild==0
   FileWrite(myfile,"            X        NONE")
else
   while greatgrandchild
       greatGrandCount=greatGrandCount+1
       workwnd=greatgrandchild
       gosub gettextandclass
       FileWrite(myfile,strcat("            ",strfix(greatGrandCount,"
",9),strfix(class," ",26),strfix(ident," ",7),text))

       greatgrandchild=cWndInfo(workwnd,6)
       gosub processlevel5
   endwhile
endif

return

:ProcessLevel5
L5child=cWndInfo(workwnd,8)
L5Count=0
if l5child==0
   FileWrite(myfile,"               X    NONE")
else
   while l5child
       l5count=L5Count+1
       workwnd=l5child
       gosub gettextandclass
       FileWrite(myfile,strcat("               ",strfix(L5Count,"
",5),strfix(class," ",26),strfix(ident," ",7),text))

       l5child=cWndInfo(workwnd,6)
       gosub processlevel5
   endwhile
endif

return


:GETTEXTANDCLASS
text=cWndInfo(workwnd,0)
ident=cWndInfo(workwnd,1)
class=cWndInfo(workwnd,2)
return
```

Partial Window Names

Alphabetical: Function List

Control Type: Function List

When using partial window names as parameters, you can specify the full name if you wish, but in most circumstances, it isn't necessary. Specify enough characters so that the "partial-windowname" matches only one existing window. If it matches more than one window, the most recently accessed window which it matches will be used. The windowname "" may be used as a shorthand way of referring to the WIL parent application window. Remember that the case (upper or lower) of the title is significant. If the case is not correct, a match will not be made.

There are a total of four distinct ways to define a partial window name. Decide which option is best for your particular needs.

Window Name

A partial windowname can be the first few defining characters of a window title. For example, WinShow("Notepad") would match any window whose title begins with "Notepad".

Exact Match

Those WIL functions which take a partial windowname as a parameter can be directed to accept only an exact match by ending the window name with a tilde (~).

A tilde (~) used as the last character of the window name indicates that the name must match the window title through to the end of the title. For example, WinShow("Note~") would only match a window whose title was "Note"; it would not match "Notepad".

Match Any Window

A tilde (~) used as the first character of the window name, will match any window containing the specified string anywhere in its title. For example, WinShow("~Notepad") will match a window title of "(Untitled) - Notepad" and a window title of "My Notepad Application", as well as a window title of "Notepad - (Untitled)".

Window Name Ends With…

A tilde (~) used as the first and last character of the window name, will match any window containing the specified string anywhere in its title, except at the beginning. For example, WinShow("~Notepad~") will match a window title of "Notepad" and a window title of "(Untitled) -Notepad", but will not match a window title of "Notepad - (Untitled)".

Alphabetical: Function List

The following WIL functions are useful when using WIL extenders.

AddExtender(filename)
Installs a WIL extender Dll.

LastError( )
Returns the last error encountered.

The following functions can be added with the WWCTL34i.DLL using the AddExtender function.

```
AddExtender("WWCTL34i.DLL")
```


- Contacting Wilson Windowware
- How to get Technical Support
- Window Analysis script
- Control Type: Function List

cCheckBox
Examines or sets a check box.

cClearLBAll
Remove the highlighting from all items in either a Single-selection or a Multiple-select List Box.

cClearLVItem
You can use this function to un-select an item of a List View common control. It may be necessary to use this function when a control allows multiple selections but only one is desired. The class name for this control is "SysListView32".

cClearTVItem
You can use this function to un-select an item of a Tree View common control. Placing the text of an item and each of its ancestor items starting at the root into a string specifies an item. A tab character must delimit each items text. The class name for this control is "SysTreeView32".

cClickButton
Clicks a button, emulating a user.

cClickToolbar
Simulates performing a left mouse button click of a Tool Bar button.

cDblClickItem
You can use this function to simulate a mouse double click on a List View, Tree View or Header common control.

cEnableState
Examines or sets the enabled/disabled status of a control.

cFindbyClass
Gets the window handle of the first window whose class name matches the specified "class-name".

cFindbyName
Gets the window handle of the first window whose window name matches the specified 'partial window name'.

cGetCalDate
Retrieve the date or date range of a Month Calendar control with this function.

cGetCBCount
Gets a count of items in a Combo Box control.

cGetCBText
Extracts the text from a drop-down combo box.

cGetControlImageCrc
Creates a Cyclic Redundancy Check (CRC) value of the on-screen image of a Windows control.

cGetDTPDate
This function returns the date and time stored in a Date Time Picker control in standard datetime format.

cGetEditText
Retrieves the contents of an Edit Box.

cGetHRText
The function will return all the text associated with each item of a Header common control. Header control items are often found as column titles of report (detail) style List View controls. Double clicking a Header item will often reverse the sort order of the column below it. The class name for this control is "SysHeader32".

cGetInfo
This function returns extender specific information.

cGetIpAddr
This function retrieves an IP address stored in an IP Address common control. The class name for this control is "SysIPAddress32".

cGetLBCount
Get a count of items in a List Box control.

cGetLBText
Extracts the text from a standard list box.

cGetLBSelText
Retrieve the text from all highlighted items in either a Single-selection or a Multiple-select List Box control.

cGetLVColText
Retrieves all the text associated with a column of a List View common control with the Report View style.

cGetLvDdtText
Returns the text associated with the item, that is the current drag and drop target of a List View common control.

cGetLvFocText
Retrieve the text associated with the items that have the focus in a List View common control.

cGetLvSelText
Retrieves the text associated with selected items of a List View common control.

cGetLVText
This function retrieves the text associated with items of a List View common control. A tab character is inserted between each item's text. If the control has the Report View style, only the text from the first column of each row is returned. The class name for this control is "SysListView32".

cGetSBText
This function returns the text that may be associated with each part of a Status Bar control. A tab character delimits each parts text. If a part does not have text, a single tab character is placed in the appropriate position of the return list. If a Status Bar is in simple mode, only the text associated with that mode will be returned. The class name for this control is "msctls_statusbar32".

cGetTBText
Retrieves the text for each button of a Tool Bar control.

cGetTrackMax
Get the maximum position of a Track Bar control.

cGetTrackMin
Get the minimum position of a Track Bar control.

cGetTrackPos
Get the current position of a Track Bar control.

cGetUpDownMax
This function allows you to get the maximum position of an UpDown control. The class name for this control is "msctls_updown32.

cGetUpDownMin
This function allows you to get the minimum position of an UpDown control. The class name for this control is "msctls_updown32.

cGetUpDownPos
Gets the position of an UpDown control.

cGetWndCursor
Determines an application windows current mouse cursor.

cPostButton
Clicks a button, emulating a user. This uses an undocumented method, different from cClickButton, that seems to work when cClickButton causes the script to hang.

cPostMessage
The real PostMessage (or PostMessageA if you want to get nitpicky).

cRadioButton
Examines or sets a radio button.

cSendMessage
The real SendMessage (or SendMessageA if you want to get nitpicky).

cSetCalDate
Set the date or date range of a Month Calendar control with this function.

cSetCBItem
Sets a single item in a standard Combobox.

cSetDTPDate
Change the values of a Date Time Picker control.

cSetEditText
Changes the contents of an Edit Box.

cSetFocus
This function sets the keyboard focus to the control window specified by the input window handle.

cSetIpAddr
This function places an IP address stored in an IP Address common control. The class name for this control is "SysIPAddress32".

cSetLBItem
Sets a single item in a standard single item list box. It will not work with a multiple selection listbox.

cSetLBItemEx
Select one or more items in a List Box control.

cSetLVItem
You can use this function to make an item of a List View common control the selected item. Once an item is selected, it is the ultimate recipient of mouse clicks and key strokes sent to the control. The class name for this control is "SysListView32".

cSetTABItem
Selects a tab from a tabbed dialog.

cSetTrackPos
Sets the position of a Track Bar control.

cSetTVItem
This function is used to select an item of a Tree View common control. Once an item is selected, it is the ultimate recipient of mouse clicks and keystrokes sent to the control. Placing the text of an item and each of its ancestor items starting at the root into a string specifies an item. A tab character must delimit each items text. The class name for this control is "SysTreeView32".

cSetUpDownPos
The function can be used to set the position of an UpDown control.

cSetWndText
Change the text associated with a Windows control.

cWinIdConvert
Convert between a Windows handle and a Window Id (pseudo-handle)

cWndByClass
Obtains the handle of a child window or control based on the window or control class.

cWndByID
Obtains the handle of a child window or control based on the window or control ID number.

cWndByName
Obtains the handle of a child window or control based on the window or control title.

cWndBySeq
Obtains the handle of a child window or control based on the window or control sequence number.

cWndByWndSpec

Returns a window handle based on the window specifications.

cWndExist

Determines if a child window of the specified parent exists.

cWndGetWndSpec

Creates a window specification from the window handle.

cWndInfo

This function returns information about a window or control.

cWndState

Get the current state of a window

Control Type: Function List

The following WIL functions are useful when using WIL extenders.

AddExtender(filename)
Installs a WIL extender Dll.

LastError( )
Returns the last error encountered.

The following functions can be added with the WWCTL34i.DLL using the AddExtender function.

```
AddExtender("WWCTL34i.DLL")
```


- Contacting Wilson Windowware
- How to get Technical Support
- Window Analysis script
- Alphabetical: Function List

Most Controls

cClickButton
Clicks a button, emulating a user.

cEnableState
Examines or sets the enabled/disabled status of a control.

cFindbyClass
Gets the window handle of the first window whose class name matches the specified "class-name".

cFindbyName
Gets the window handle of the first window whose window name matches the specified 'partial window name'.

cPostButton
Clicks a button, emulating a user. This uses an undocumented method, different from cClickButton, that seems to work when cClickButton causes the script to hang.

cSetWndText
Change the text associated with a Windows control.

cWinIdConvert
Convert between a Windows handle and a Window Id (pseudo-handle)

cWndByClass
Obtains the handle of a child window or control based on the window or control class.

cWndByID
Obtains the handle of a child window or control based on the window or control ID number.

cWndByName
Obtains the handle of a child window or control based on the window or control title.

cWndBySeq
Obtains the handle of a child window or control based on the window or control sequence number.

cWndExist
Determines if a child window of the specified parent exists.

cWndInfo
This function returns information about a window or control.

cWndState
Get the current state of a window

CHECKBOX

cCheckBox
Examines or sets a check box.

COMBOBOX

cGetCBCount
Gets a count of items in a Combo Box control.

cGetCBText
Extracts the text from a drop-down combo box.

cSetCBItem
Sets a single item in a standard Combobox.

EDITBOX

cGetEditText
Retrieves the contents of an Edit Box.

cSetEditText
Changes the contents of an Edit Box.

LISTBOX

cClearLBAll
Remove the highlighting from all items in either a Single-selection or a Multiple-select List Box.

cGetLBCount
Get a count of items in a List Box control.

cGetLBText
Extracts the text from a standard list box.

cGetLBSelText
Retrieve the text from all highlighted items in either a Single-selection or a Multiple-select List Box control.

cSetLBItem
Sets a single item in a standard single item list box. It will not work with a multiple selection listbox.

cSetLBItemEx
Select one or more items in a List Box control.

msctls_statusbar32

cGetSBText
This function returns the text that may be associated with each part of a Status Bar control. A tab character delimits each parts text. If a part does not have text, a single tab character is placed in the appropriate position of the return list. If a Status Bar is in simple mode, only the text associated with that mode will be returned. The class name for this control is "msctls_statusbar32".

msctls_trackbar32

cGetTrackMax
Get the maximum position of a Track Bar control.

cGetTrackMin
Get the minimum position of a Track Bar control.

cGetTrackPos
Get the current position of a Track Bar control.

cSetTrackPos
Sets the position of a Track Bar control.

msctls_updown32

cGetUpDownMax
This function allows you to get the maximum position of an UpDown control. The class name for this control is "msctls_updown32.

cGetUpDownMin
This function allows you to get the minimum position of an UpDown control. The class name for this control is "msctls_updown32.

cGetUpDownPos
Gets the position of an UpDown control.

cSetUpDownPos
The function can be used to set the position of an UpDown control.

MOUSE

cGetWndCursor
Determines an application windows current mouse cursor.

PUSHBUTTON

cClickButton
Clicks a button, emulating a user.

cPostButton
Clicks a button, emulating a user. This uses an undocumented method, different from cClickButton, that seems to work when cClickButton causes the script to hang.

RADIOBUTTON

cRadioButton
Examines or sets a radio button.

STATIC

cWndInfo
This function returns information about a window or control.

SysDateTimePick32

cGetDTPDate
This function returns the date and time stored in a Date Time Picker control in standard datetime format.

cSetDTPDate
Change the values of a Date Time Picker control.

SysHeader32

cDblClickItem
You can use this function to simulate a mouse double click on a List View, Tree View or Header common control.

cGetHRText
The function will return all the text associated with each item of a Header common control. Header control items are often found as column titles of report (detail) style List View controls. Double clicking a Header item will often reverse the sort order of the column below it. The class name for this control is "SysHeader32".

SysIPAddress32

cGetIpAddr

This function retrieves an IP address stored in an IP Address common control. The class name for this control is "SysIPAddress32".

cSetIpAddr
This function places an IP address stored in an IP Address common control. The class name for this control is "SysIPAddress32".

SysListView32

cClearLVItem
You can use this function to un-select an item of a List View common control. It may be necessary to use this function when a control allows multiple selections but only one is desired. The class name for this control is "SysListView32".

cDblClickItem
You can use this function to simulate a mouse double click on a List View, Tree View or Header common control.

cGetLVColText
Retrieves all the text associated with a column of a List View common control with the Report View style.

cGetLvDdtText
Returns the text associated with the item, that is the current drag and drop target of a List View common control.

cGetLvFocText
Retrieve the text associated with the items that have the focus in a List View common control.

cGetLvSelText
Retrieves the text associated with selected items of a List View common control.

cGetLVText
This function retrieves the text associated with items of a List View common control. A tab character is inserted between each item's text. If the control has the Report View style, only the text from the first column of each row is returned. The class name for this control is "SysListView32".

cSetLVItem
You can use this function to make an item of a List View common control the selected item. Once an item is selected, it is the ultimate recipient of mouse clicks and key strokes sent to the control. The class name for this control is "SysListView32".

SysMonthCal32

cGetCalDate
Retrieve the date or date range of a Month Calendar control with this function.

cSetCalDate
Set the date or date range of a Month Calendar control with this function.

SysTabControl32

cSetTABItem
Selects a tab from a tabbed dialog.

SysTreeView32

cDblClickItem
You can use this function to simulate a mouse double click on a List View, Tree View or Header common control.

cClearTVItem
You can use this function to un-select an item of a Tree View common control. Placing the text of an item and each of its ancestor items starting at the root into a string specifies an item. A tab character must delimit each items text. The class name for this control is "SysTreeView32".

cSetTVItem
This function is used to select an item of a Tree View common control. Once an item is selected, it is the

ultimate recipient of mouse clicks and keystrokes sent to the control. Placing the text of an item and each of its ancestor items starting at the root into a string specifies an item. A tab character must delimit each items text. The class name for this control is "SysTreeView32".

ToolbarWindow32

cClickToolbar
Simulates performing a left mouse button click of a Tool Bar button.

cGetTBText
Retrieves the text for each button of a Tool Bar control.

Non-Control Related Functions

cGetControlImageCrc
Creates a Cyclic Redundancy Check (CRC) value of the on-screen image of a Windows control.

cGetInfo
This function returns extender specific information.

cGetWndCursor
Determines an application windows current mouse cursor.

cPostMessage
The real PostMessage (or PostMessageA if you want to get nitpicky).

cSendMessage
The real SendMessage (or SendMessageA if you want to get nitpicky).

cSetFocus
This function sets the keyboard focus to the control window specified by the input window handle.

cWndExist
Determines if a child window of the specified parent exists.

cWndByWndSpec
Returns a window handle based on the window specifications.

cWndGetWndSpec
Creates a window specification from the window handle.

cCheckBox

This function is used to examine or modify the status of checkboxes.

Syntax:

cCheckBox(hwnd, flag)

Parameters:

(i) hwnd	CheckBox handle from a cWndBy... function.
(i) flag	See below.

Returns:

(i)	Previous state of checkbox.

Flags

-1	Examine the settings of a checkbox 0 Not Checked 1 Checked 2 Grayed (not disabled) (checks status of a 3-state checkbox)
0	Clears a checkbox
1	Sets a checkbox
2	Grays a 3-state checkbox (Do NOT use on standard checkboxes)

Example:

```
;Note:  This example designed for Windows 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL mmsys.cpl")

multiwnd=DllHwnd("Multimedia Properties")
audiownd=cWndByName(multiwnd,"Audio")
prefchkbox=cWndByname(audiownd,"Use preferred devices only")
chk=cCheckbox(prefchkbox,-1)
Message("Use Preferred Devices is set to",chk)
exit
```

See Also:

cRadioButton, cClickButton

cClearLBAll

Remove the highlighting from all items in either a Single-selection or a Multiple-select List Box.

Syntax:

cClearLBAll (window-handle)

Parameters:

(i) window-handle	Windows handle to a List Box control.

Returns:

(i)	@TRUE, If all items are unselected @FALSE, If one or more items are still selected.

Use this function to remove the highlighting from all items in either a Single-selection or a Multiple-select List Box. It is a good idea to call this function before calling cSetLBItemEX on Multiple-select list boxes because the list box may have some items already selected.

If you pass in a handle to a list box that does not have any items selected **cClearLBAll** will not create an error and will return @TRUE.

The class name for List Box controls is "List Box.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

;Get the handle of the parent window
;of the list box control.
HwndParent=DllHwnd("Window Title")

hwnd = cwndbyclass(hwndParent, "ListBox")

; Clear any current selections
if !cClearLBAll(hwnd)
    goto ErrorHandler
endif

; Select the first and fifth items in the list
if ! cSetLBItemEX(hwnd, 1 5)
   goto ErrorHandler
endif

; Get the selected items text
SelectedText = cGetLBSelText(hwnd)

SelectedText = strreplace(SelectedText, @TAB, @CRLF)
message("Selected Text", SelectedText)
```

See Also:

cGetLBText, cSetLBItem, cGetLBSelText, cSetLBItemEX

cClearLVItem

You can use this function to un-select an item of a List View common control. It may be necessary to use this function when a control allows multiple selections but only one is desired.

The class name for this control is "SysListView32".

Syntax:

cClearLVItem (window-handle, item)

Parameters:

(i) window-handle	Windows handle to a List View common control.
(i) item	The one based number of the item to un-select. I.e., the item's location in a list whose first item is item #1.

Returns:

(i)	@TRUE, if item is not selected. @FALSE, if item could not be un-selected.

Example:

```
;  Add the control manager extender.     HERE??!!
AddExtender("wwctl34i.dll")

; Explorer List View area
hwndex = DLLhwnd("Exploring ")
hwnd   = cwndbyclass(hwndex,"SHELLDLL_DefView")
hwnd   = cwndbyclass(hwnd,"SysListView32")

; Get the labels from the first column of each row in the control.
wintext =  cgetlvtext(hwnd)

; Clear all items so.
nItems = itemcount(wintext, @TAB)
for i=1 to nItems

; Unselect the item.
    cclearlvitem(hwnd, i)
next
```

See Also:

cGetLVText, cSetLVItem, cDblClickItem, cWndByClass, cWndinfo, DLLhwnd (see WIL help).

cClearTVItem

You can use this function to un-select an item of a Tree View common control. Placing the text of an item and each of its ancestor items starting at the root into a string specifies an item. A tab character must delimit each items text.

The class name for this control is "SysTreeView32".

Syntax:

cClearTVItem (window-handle, item-path)

Parameters:

(i) window-handle	Windows handle to a List View common control.
(s) item-path	Tab-delimited list of item text.

Return:

(i)	@TRUE, if item is not selected. @FALSE, if item could not be un-selected.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find the Explorer tree control
hwnd      = DLLhwnd("Exploring")
hwnd      = cwndbyclass(hwnd, "BaseBar")
hwnd      = cwndbyclass(hwnd, "ReBarWindow32")
hwnd      = cwndbyclass(hwnd,"SysTreeView32")

;Remove traces of our previous activities.
if ccleartvitem(hwnd, selection)

   ; Report success.
   selection = strreplace(selection,@TAB, @CRLF)
   message("Clear Item",  "Node tree:%@CRLF% %selection% %@CRLF%- cleard")
else

   ; Can't find the item
   selection = strreplace(selection,@TAB, @CRLF)
   message("Clear Item",  "Node tree:%@CRLF% %selection% %@CRLF%- not cleard")
endif
```

See Also:

cSetTVItem, cDblClickItem, cWndByClass, cWndInfo, DLLhwnd,

cClickButton

Clicks a button, emulating a user.

Syntax:

cClickButton(hwnd)

Parameters:

(i) hwnd	PushButton handle from a cWndBy... function.

Returns:

(0)	always 0.

This function is designed to click a push button. However it can also be used to click most types of controls that can accept a button click. This function uses the recommended procedures to push the button. However is some cases the script will appear to hang. If your script seems to hang after a **cClickButton** call, try the **cPostButton** function instead.

Example:

```
AddExtender("wwctl34i.dll")
Run("notepad.exe","")
SendMenusTo("~Notepad","FileOpen")
WinWaitExist("Open",5)
openwnd=DllHwnd("Open")
editwnd=cWndByID(openwnd,1152)
mask=cSetEditText(editwnd, "*.*")
openbutton=cWndByName(openwnd,"Open")
cClickButton(openbutton)
Message("Open file mask changed to *.*","and Open button clicked.")
exit
```

See Also:

cPostButton, cCheckBox, cRadioButton

cClickToolbar

Simulates performing a left mouse button click of a Tool Bar button.

Syntax:

cClickToolbar(window-handle, position)

Parameters:

(i) window-handle	Windows handle to a Tool Bar common control.
(i) position	Position of button to click.

Return:

(i)	@TRUE, if button pressed @FALSE, if button was not pressed.

You can simulate performing a left mouse button click of a Tool Bar button with this function. The button is selected by giving its one-based position as the second parameter. I.e., specifying the item's location in a list whose first item is item #1. Since buttons can be hidden, a button's position can not always be determined by visual inspection. If a button has associated text, the position can be determined with the cGetTbText function.

The class name for the Tool Bar control is "ToolbarWindow32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find the Explorer tool bar handle
hwndex     = DLLhwnd("Exploring ")
hwnd       = cwndbyclass(hwndex,"Worker")
hwnd       = cwndbyclass(hwnd,"ReBarWindow32")
hwnd       = cwndbyclass(hwnd,"ToolbarWindow32")
buttontext = "Forward"

; Get any button text
alltext = cgettbtext(hwnd)

;Get the index of the button
select1=ItemLocate(buttontext, alltext, @tab)

; Get any button text
alltext = cgettbtext(hwnd)

;Get the index of the button
select1=ItemLocate(buttontext, alltext, @tab)

if  select1==0

   Message("Tool Bar Button","%buttontext% Not Found")
else
```

```
    ; Click the Forward button
    if cclicktoolbar(hwnd, select1)
          message("Button click", "Succeeded")
    else
          message("Button click", "Failed")
    endif
endif
```

See Also:

cGetTBText, cWndByClass, cWndInfo, DLLhwnd,

cDblClickItem

You can use this function to simulate a mouse double click on a List View, Tree View or Header common control.

Syntax:

cDblClickItem (window-handle, item)

Parameters:

(i) window-handle	Windows handle to a List View common control.
(i) item	The one-based number of the item to double click. I.e., the item's location in a list whose first item is item #1 (rather than 0).

Returns:

(i)	@TRUE, if item is not selected. @FALSE, if item could not be un-selected.

The second parameter to this function operates differently based on the class of the target control.

For List View controls, the parameter is optional. If it is a number greater than zero, the function will attempt to double click the item associated with the number. If the second parameter is zero or less, it will double click a previously selected item on the List View control.

For Tree View controls, the parameter is ignored because there is no numerical association made with the control's items. Use the cSetTvItem function to select a Tree View control item before calling this routine to simulate the double click.

For Header controls, the second parameter must be specified. This is because Header controls do not allow individual items to be pre-selected the way that List View and Tree View controls do. The table below illustrates these differences.

Control Class Name	Second Parameter
SysListView32	Optional
SysTreeView32	Ignored
Header32	Mandatory

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Explorer List View area
hwndex = DLLhwnd("Exploring ")
hwnd   = cwndbyclass(hwndex,"SHELLDLL_DefView")
```

```
hwnd   = cwndbyclass(hwnd,"SysListView32")
myitem = "Backup"

; Get the labels from the first column of each row in the control.
wintext = cgetlvtext(hwnd)

;Get the index of the item
select1 = ItemLocate(myitem, wintext, @tab)

if select1 > 0

cDblClickItem(hwnd, select1)
endif
```

See Also:

cWndByClass, cWndInfo, DLLhwnd

cEnableState

Examines or sets the enabled/disabled status of a control.

Syntax:

cEnableState(hwnd, flag)

Parameters:

(i) hwnd	Control handle from a cWndBy... function.
(i) flag	See below.

Returns:

(i)	Previous state of control.

Flags

-1	Examine the state of a control
	0 Disabled
	1 Enabled
0	Disables a control
1	Enables a control

This function is used to examine or modify the state of controls. Whether or not the application owning the control can make sense of a modified control is beyond the scope of this function.

Example:

```
;Note:  This example designed for Windows 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL mmsys.cpl")

multiwnd=DllHwnd("Multimedia Properties")
audiownd=cWndByName(multiwnd,"Audio")
prefchkbox=cWndByname(audiownd,"Use preferred devices only")
chk=cEnableState(prefchkbox,0)
Message("Use Preferred Devices","Has been disabled")
chk=cEnableState(prefchkbox,1)
Message("Use Preferred Devices","Has been enabled")
exit
```

See Also:

cRadioButton, cCheckBox, cGetEditText

cFindbyClass

Gets the window handle of the first window whose class name matches the specified "class-name".

Syntax:

cFindbyClass(class-name)

Parameters:

(s) class-name	Window class name.

Returns:

(i) window-handle	Windows handle.

cFindByClass returns the windows handle of the first window whose class name matches the specified "class-name". It can be used to manipulate windows that do not have titles. The function searches by checking each top-level windows class. If no match is found, it checks the child windows of each top-level window. 0 is returned, if no window has the class name.

Note: "Class-name" is not case sensitive but must be the full class name (ie, not a partial name).

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
sAppName     = "Notepad.exe"
sAppWindow = "~Notepad"
sAppClass      = "NotePad"
; Start up an application
TestApp = FileLocate (sAppName)
if TestApp == "" then goto Error
if !ShellExecute(TestApp, "", "",0, "") then goto errorif !WinWaitExist(sAppWindow,
8)  then goto error

hWnd  = cFindbyClass(sAppClass)
if !hWnd then goto Error
; Get the first edit control we can find.
sClassName = "Edit"
hWndEdit    =  cFindbyClass(sClassName)
; Get Notepad's edit control
hWndEdit2 = cWndByClass(hWnd, sClassName)
if hWndEdit == hWndEdit2
   message("Edit Controls", "Found Notepad's edit with cFindbyClass")
else
   message("Edit Controls", "Found another applications edit control")
endif
exit
:Error
; Do something useful.
```

See Also:

cWndByClass

cFindbyName

Gets the window handle of the first window whose window name matches the specified 'partial window name'.

Syntax:

cFindbyName(partial-name)

Parameters:

(s) partial-name	Partial window name.

Returns:

(i) window-handle	Windows handle.

cFindByName returns the windows handle of the first window whose name matches the specified "partial-name". All top-level and child windows are searched. 0 is returned, if no window includes the partial window name in its title.

See Partial Window Names for a detailed explanation of the rules governing, partial window names.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find a button handle
sAppWndName   = "About YA"

hWnd = cFindByName(sAppWndName)
hWnd = cWndByName(hWnd, "Evaluate")
if !cClickButton( hWnd )
   exit
endif
```

See Also:

DllHwnd, cWndByName

cGetCalDate

Retrieves the date or date range of a Month Calendar control.

Syntax:

cGetCalDate (window-handle)

Parameters:

(i) window-handle	Windows handle to a Month Calendar common control.

Return:

(s) list	Date or date range in date-time format.

You can retrieve the date or date range of a Month Calendar control with this function. Month Calendar controls may or may not allow a date range to be specified. If the control you are retrieving allows the selection of a range of dates, the return string will be a tab-delimited list of the minimum and maximum dates. If the control does not allow range selection, the return string will contain a single date. In either case, the dates will be in the standard date-time format.

The class name for the Month Calendar control is "SysMonthCal32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
; Get a handle
hwndex = DLLhwnd("Microsoft Control Spy")
hwnd   = cwndbyclass(hwndex,"CSPYContainer")
hwnd   = cwndbyclass(hwnd,"SysMonthCal32")
; Get the time set in the control.
ctldates = cgetcaldate( hwnd )
if strlen(ctldates)
   ;Does this control have a range.
   count =ItemCount(ctldates, @tab)
   if count == 2
      ;Get each time
      startdate = ItemExtract(1, ctldates, @TAB)
      enddate   = ItemExtract(2, ctldates, @TAB)
      ; Are they the same date?
      if timediffdays(startdate, enddate) == 0
        dates = startdate
      else
        dates = "From %startdate% to %enddate%."
      endif
      message("Control date range", dates)
   else
      message("Control date", startdate)
   endif
endif
```

See Also:

cSetCalDate, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetCBCount

Gets a count of items in a Combo Box control.

Syntax:

cGetCBCount (window-handle)

Parameters:

(i) window-handle	window handle to a Combo Box control.

Return:

(i) count	number of items in Combo Box.

Note: Get the handle of the parent window of the Combo Box control.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
hwnd = cwndbyclass(hwndParent, "ComboBox")

; Get the number of items in the List Box.
nCount = cGetCBCount(hwnd)

message("Number of items in combo box", nCount)
```

See Also:

cSetCBItem

cGetCBText

Extracts the text from a drop-down combo box.

Syntax:

cGetCBText(hwnd)

Parameters:

(i) hwnd	ComboBox handle from a cWndBy... function.

Returns:

(s)	Combobox contents in a tab-delimited list.

This function is designed to extract the contents of a combobox, presumably so the contents may be examined to determine which item to set.

Example:

```
;Note:  This example designed for Windows 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

netwnd=DllHwnd("Network")
cfghwnd= cWndByName(netwnd,"Configuration")
cbWnd=cWndByID(cfghwnd,405)
response=cGetCBText(cbwnd)
response=strreplace(response,@tab,@crlf)
Message("Response",response)
exit
```

See Also:

cSetCBItem, cGetEditText, cGetLBText

cGetControlImageCrc

Creates a Cyclic Redundancy Check (CRC) value of the on-screen image of a Windows control.

Syntax:

cGetControlImageCrc(window-handle)

Parameters:

(i) window-handle	window handle of control to create CRC value for.

Returns:

(i) image-crc	CRC value of control image.

Use this function to create a Cyclic Redundancy Check (CRC) value of the on-screen image of a Windows control. Different CRCs can be obtained for different visual states of a control. The CRCs can be saved and later used to determine the current state of the control by comparing the control's current CRC with the saved CRCs.

Note: Cyclic Redundancy Check is an error-checking technique in which a remainder is calculated by dividing binary content by a prime binary divisor. Differences (errors) are detected by comparing the calculated remainder to a stored remainder.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll ")

;Get ini file name
moi=IntControl(1004,0,0,0,0)
moipath=Filepath(moi)
inifile=strcat(moipath,"test.ini")
run("notepad.exe","")

hNoteWnd=cWndByWndSpec("Notepad","NOTEPAD",1,15)
cSetFocus(hNoteWnd)     ;Selects target window to receive keystrokes
TimeDelay(1)
SendKey(`!fo`)     ;Sends Keystrokes

window1=cWndByWndSpec("#32770","NOTEPAD",13,1091,1137,1088,1120,1121,10
90,1152,1089,1136,1040,1,2,1038)
ControlHandle=cWndByName(window1,`Open~`)

; So if we assume that we have no idea about the "Open" button, as to
; whether it is disables or green or whatever…

while 1
   TimeDelay(2)
    crc=cGetControlImageCRC(ControlHandle)

    ;What is it
    ;Assume options are "normal" "disabled" "green"
    state=IniReadPvt("DaOpenButton",crc,"unknown",inifile)
```

```
    if state=="unknown"
      list="normal disabled green"
      state=AskItemList("What state is the open button in? %crc%",list,"
",@sorted,@single)
      IniWritePvt("DaOpenButton",crc,state,inifile)
    Endif

    Pause("Button State",state)
endwhile
```

See Also:

cWndState

cGetDTPDate

This function returns the date and time stored in a Date Time Picker control in standard datetime format.

Syntax:

cGetDTPDate (window-handle)

Parameters:

(i) window-handle	Windows handle to a Date Time Picker common control.

Return:

(s) date-time	Time in YYYY:MM:DD:HH:MM:SS format.

This function returns the date and time stored in a Date Time Picker control in standard "datetime" format. The output can be passed directly to any WIL time function that accepts the standard YMDHMS format. All parts of the standard format will be returned even if the control only displays partial information.

The class name for the Date Time Picker control is "SysDateTimePick32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Get a handle
hwndex = DLLhwnd("Microsoft Control Spy")
hwnd   = cwndbyclass(hwndex,"CSPYContainer")
hwnd   = cwndbyclass(hwnd,"SysDateTimePick32")

; Get the time set in the control.
ctldatetime = cgetdtpdate( hwnd )

;subract a year and a second.
ctldatetime = timesubtract(ctldatetime, "01:00:00:00:00:01")

if csetdtpdate(hwnd, ctldatetime)
ctldatetime = cgetdtpdate( hwnd )

message(" New DateTime", ctldatetime)
endif
```

See Also:

cSetDTPDate, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetEditText

Retrieves the contents of an Edit Box.

Syntax:

cGetEditText(hwnd)

Parameters:

(i) hwnd	EditBox handle from a cWndBy… function.

Returns:

(s)	Contents of the editBox.

This function is used to examine the contents of edit controls.

Example:

```
AddExtender("wwctl34i.dll")
Run("notepad.exe","")
SendMenusTo("~Notepad","FileOpen")
WinWaitExist("Open",5)
openwnd=DllHwnd("Open")
editwnd=cWndByID(openwnd,1152)
mask=cGetEditText(editwnd)
Message("Open file mask is",mask)
exit
```

See Also:

cSetEditText, cGetLBText, cGetCBText

cGetHRText

The function will return all the text associated with each item of a Header common control. Header control items are often found as column titles of report (detail) style List View controls. Double clicking a Header item will often reverse the sort order of the column below it.

The class name for this control is "SysHeader32".

Syntax:

cGetHRText(window-handle)

Parameters:

(i) window-handle	Windows handle to a List View common control.

Return:

(s) list	A tab-delimited list of the text associated with each item

Example:

```
;  Add the control manager extender dll.
AddExtender("wwctl34i.dll")

;Find the Outlook Express header control
hwnd     = DLLhwnd("Inbox - Outlook")
hwnd     = cwndbyclass(hwnd,"ThorCmnViewWndClass")
hwnd     = cwndbyclass(hwnd,"ThorMsgViewWndClass")
hwnd     = cwndbyclass(hwnd,"SysListView32")
hwnd     = cwndbyclass(hwnd,"SysHeader32")
mycolumn = "Subject"

; Get the text for each item of the Header control.
sbtext = cgethrtext(hwnd)

;Get the index of the item of interest
select1=ItemLocate(mycolumn, sbtext, @tab)

if select1 < 1

   Message("Header column - %mycoumn%","Not Found")
else
   ; Reverse the sort order of the in box messages.
   cdblclickitem(hwnd, select1)
endif
```

See Also:

cDblClickItem, cWndByClass, cWndinfo, DLLhwnd (WIL help).

cGetInfo

This function returns extender specific information.

Syntax:

cGetInfo(request)

Parameters:

(i) request	Request code for desired information (see below).

Returns:

(s)	Desired Information.

Request

0	Extender version number
1	Total number of defined functions in extender
2	Total number of defined constants in extender

Example:

```
AddExtender("wwctl34i.dll")
ver=cGetInfo(0)
Message("Extender Version Number",ver)
```

See Also:

cWndInfo

cGetIpAddr

This function retrieves the IP address stored in an IP Address common control.

The class name for this control is "SysIPAddress32".

Syntax:

cGetIpAddr(window-handle)

Parameters:

(i) window-handle	Windows handle to an IP Address Controls.

Returns:

(s) IP Address	An Internet protocol (IP) format address.

Note: Several Windows system property sheets contain controls that look and behave like an IP Address control. However, these controls are not a common control and have different class names. This function does make an attempt to identify and extract an IP address from these pseudo controls.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find Control Spy ip address window handle.
;hwnd = DLLhwnd("Microsoft Control Spy -")
;hwnd = cwndbyclass(hwnd, "CSPYContainer")
;hwnd = cwndbyclass(hwnd,"SysIPAddress32")

; Get the address.
ipaddr = cgetipaddr(hwnd)

; Get the window class name and display the new IP address.
wintext = cwndinfo(hwnd, "2")
if wintext == ""
   wintext = "a Classless Window"
endif
message("IP Address from %wintext%.", "IP Address: %ipaddr%.")
```

See Also:

cSetIpAddr, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetLBCount

Get a count of items in a List Box control.

Syntax:

cGetLBCount (window-handle)

Parameters:

(i) window-handle	window handle to a List Box control.

Returns:

(i) count	number of items in List Box

Note: Use this function to get a count of items in a List Box control. The function can obtain the count for "Owner Drawn" and non-text item List Boxes, as well as, conventional List Boxes.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
hwnd = cwndbyclass(hwndParent, "ListBox")

; Clear any current selections
if !cClearLBAll(hwnd)
    goto ErrorHandler
endif

; Get the number of items in the List Box.
nCount = cGetLBCount(hwnd)

; Select the last item in the list box
if ! cSetLBItemEX(hwnd, nCount)
   goto ErrorHandler
endif

; Get the selected items text
SelectedText = cGetLBSelText(hwnd)

message("Last Items Text", SelectedText)
```

See Also:

cGetLBText, cSetLBItem, cGetLBSelText, cSetLBItemEX, cClearLBAll

cGetLBText

Extracts the text from a standard list box.

Syntax:

cGetLBText(hwnd)

Parameters:

(i) hwnd	ListBox handle from a cWndBy... function.

Returns:

(s)	Listbox contents in a tab delimited list.

This function is designed to extract the contents of a listbox, presumably so the contents may be examined to determine which item to set.

Example:

```
;Note:  This example designed for Windows 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

netwnd=DllHwnd("Network")
cfghwnd= cWndByName(netwnd,"Configuration")
lbWnd=cWndbyID(cfghwnd,404)
response=cGetLBText(lbwnd)
response=strreplace(response,@tab,@crlf)
Message("Response",response)
exit
```

See Also:

cSetLBItem, cGetEditText, cGetCBText

cGetLBSelText

Retrieve the text from all highlighted items in either a Single-selection or a Multiple-select List Box control.

Syntax:

cGetLBSelText (window-handle)

Parameters:

(i) window-handle	Windows handle to a List Box control.

Returns:

(s) item-text-list	A tab delimited list of the text associated with each selected item

Use this function to retrieve the text from all highlighted items in either a Single-selection or a Multiple-select List Box control. The text from each item is separated by a @TAB character when more than one item is highlighted. This function differs from cGetLBText. cGetLBSelText returns the text for selected items, whereas, cGetLBText returns the text from all items in the control.

Some List Boxes store text in a non standard, application specific way. **CGetLBSelText** can not retrieve text from these List Boxes and will generate a runtime error. If a List Box control does not have any items selected or does not have any items in its list, this function will return the string *EMPTY*

The class name for List Box controls is "List Box.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

;Get the handle of the parent window of the list box control.

hwnd = cwndbyclass(hwndParent, "ListBox")

; Clear any current selections
if !cClearLBAll(hwnd)
    goto ErrorHandler
endif

; Select the first and fifth items in the list
if ! cSetLBItemEX(hwnd, "1 5")
   goto ErrorHandler
endif

; Get the selected items text
SelectedText = cGetLBSelText(hwnd)

SelectedText = strreplace(SelectedText, @TAB, @CRLF)
message("Selected Text", SelectedText)
```

See Also:

cGetLBText, cSetLBItem, cClearLBAll, cSetLBItemEX

cGetLVColText

Retrieves all the text associated with a column of a List View common control with the Report View style.

Syntax:

cGetLVColText (window-handle, column-number)

Parameters:

(i) window-handle	Windows handle to a List View common control.
(i) column-number	One based number of column to extract text from.

Returns:

(s) list	A tab delimited list of the text associated with column-number for each row of the list.

This function retrieves all the text associated with a column of a List View common control with the Report View style. A tab character is inserted between each rows text from the column. If the control does not have the Report View style, the function behaves like **cGetLVText** when the second parameter is set to the number 1. The class name for this control is "**SysListView32**".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Explorer List View area
hwndex = DLLhwnd("Exploring ")
hwnd   = cwndbyclass(hwndex,"SHELLDLL_DefView")
hwnd   = cwndbyclass(hwnd,"SysListView32")
myitem = "File Folder"
mycolumn = 3

; Get the labels from the first column of each row in the control.
wintext =  cgetlvcoltext(hwnd, mycolumn)

;Get the index of the item
select1=ItemLocate(myitem, wintext, @tab)

if select1

    ; Select the item from the list control.
    if csetlvitem(hwnd, select1)
      cdblclickitem(hwnd, "")
    else
      ; Can't select the item
      message("Item not selected", "Item number %select1% may no longer exist.")
    endif
else
    message("Item not found",  "Item number %select1% may no longer exist.")
endif
```

See Also:

cGetLVText, cSetLvItem, cClearLvItem, cDblClickItem, cWndByClass, cWndinfo

cGetLvDdtText

Returns the text associated with the item, that is the current drag and drop target of a List View common control.

Syntax:

cGetLvDdtText(window-handle)

Parameters:

(i) window-handle	Windows handle to a List View common control.

Returns:

(s) text	Text associated with item that is the drag and drop target.

This function returns the text associated with the item, that is the current drag and drop target of a List View common control. An item usually becomes a drop target when the item has another item moved over the top of it. Only one item can be drop target at a time so this function will return at most one item's text. A blank string ("") is returned when no item is the drop target. The class name for the List View control is "SysListView32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
;  Loop until we find a drop target.
while @true
   TimeDelay(.5)

   ;  Get the control's handle.
   hwnd = cFindByClass("ExploreWClass")
   hwnd = cWndByClass(hwnd,"SHELLDLL_DefView")
   hwnd = cWndByClass(hwnd,"SysListView32")
   sDdtText = cGetLvDdtText(hwnd)

   ; Found a drop target.
   if strlen(sDdtText)
           message( "Drop target", sDdtText)
           break
   endif
endwhile
```

See Also:

cGetLvColText, cGetLVText, cSetLvItem, cClearLvItem, cDblClickItem, cWndByClass, cWndinfo

cGetLvFocText

Retrieve the text associated with the items that have the focus in a List View common control.

Syntax:

cGetLvFocText(window-handle)

Parameters:

(i) window-handle	Windows handle to a List View common control.

Returns:

(s) text	Text associated with item that has the focus.

Use this function to retrieve the text associated with the items that have the focus in a List View common control. Although more than one item may be selected, only one item can have the focus. The focused item is often surrounded by the standard focus rectangle. This function can return item text even when no item has the focus rectangle. This is because the control has a default item when no items have been selected. The class name for the List View control is "SysListView32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
; Get My computer's list view.
hwnd = cFindByName("~My Computer")
hwnd = cWndByClass(hwnd,"SHELLDLL_DefView")
hwnd = cWndByClass(hwnd,"SysListView32")

; Get the focused item
sFocusText = cGetLvFocText(hwnd)

if strcmp(sFocusText, "(C:)")

   ; Start a new Explorer window showing the contents of the C drive.
   sendkeysto("~My Computer", "{ENTER}")
else
   message("Focus is not on the C drive ", "It is on %sFocusText%")
endif
```

See Also:

cGetLvDdtText, cGetLvColText, cGetLVText, cSetLvItem, cClearLvItem, cDblClickItem, cWndByClass, cWndinfo

cGetLvSelText

Retrieves the text associated with selected items of a List View common control.

Syntax:

cGetLvSelText(window-handle)

Parameters:

(i) window-handle	Windows handle to a List View common control.

Returns:

(s) list	A tab delimited list of text associated with each selected item of the list

This function retrieves the text associated with selected items of a List View common control. A tab character is inserted between each items text. If the control has multiple columns, only the text from the first column is returned. The function returns a blank ("") string, if no items are selected. The class name for the List View control is " SysListView32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Explorer List View area
hwnd   = cFindByClass("ExploreWClass")
hwnd   = cwndbyclass(hwndex,"SHELLDLL_DefView")
hwnd   = cwndbyclass(hwnd,"SysListView32")

; Prime the pump.
sSelText = cGetLvSelText(hwnd)

; Loop while text is selected.
while strlen(sSelText) > 0

  ; Display current selection
  message("Selected Text", sSelText)

  ; Handles can change.
  hwnd = cFindByClass("ExploreWClass")
  hwnd = cWndByClass(hwnd,"SHELLDLL_DefView")
  hwnd = cWndByClass(hwnd,"SysListView32")

  ; Get current selection
  sSelText = cGetLvSelText(hwnd)

endwhile
```

See also:

cSetLvItem, cClearLvItem, cDblClickItem, cGetLvText, cGetLvColText

cGetLVText

This function retrieves the text associated with items of a List View common control. A tab character is inserted between each item's text. If the control has the Report View style, only the text from the first column of each row is returned.

The class name for this control is "SysListView32".

Syntax:

cGetLvText (window-handle)

Parameters:

(i) window-handle	Windows handle to a List View common control.

Returns:

(s) list	A tab-delimited list of the text associated with each item of the list.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
Run("Explorer.exe","")
WinWaitExist("Exploring",10)
; Explorer List View area
hwndex = DLLhwnd("Exploring")
hwnd   = cwndbyclass(hwndex,"SHELLDLL_DefView")
hwnd   = cwndbyclass(hwnd,"SysListView32")
myitem = "Windows"
; Get the labels from the first column of each row in the control.
wintext =  cgetlvtext(hwnd)

;Get the index of the item
select1=ItemLocate(myitem, wintext, @tab)

if select1
   ; Select the item from the list control.
   if csetlvitem(hwnd, select1)
     cdblclickitem(hwnd, "")
   else
     ; Can't select the item
     message("Item not selected", "Item number %select1% no longer exists.")
   endif
else
   message("Item not found",  "Item number %select1% may no longer exist.")
endif
```

See Also:

cSetLvItem, cClearLvItem, cDblClickItem, cWndByClass, cWndinfo, cGetLVColText, DLLhwnd (WIL help).

cGetSBText

The function returns the text that may be associated with each part of a Status Bar control. A tab character delimits each parts text. If a part does not have text, a single tab character is placed in the appropriate position of the return list. If a Status Bar is in simple mode, only the text associated with that mode will be returned.

The class name for this control is "msctls_statusbar32".

Syntax:

cGetSBText (window-handle)

Parameters:

(i) window-handle	Windows handle to a List View common control.

Returns:

(s) list	Text for all parts of a status bar control or the simple mode text.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find the Explorer status bar control
hwnd      = DLLhwnd("Exploring")
hwnd      = cwndbyclass(hwnd, "msctls_statusbar32")

; Get the text for each part of the Status Bar control.
sbtext = cgetsbtext(hwnd)

if strlen( sbtext )

   ; Display the text
   sbtext = strreplace(sbtext,@TAB, @CRLF)
   message("Status bar text",  sbtext)
else
   message("Status bar text", "Failed to obtain text.')
endif
```

See Also:

cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetTBText

Retrieves the text for each button of a Tool Bar control.

Syntax:

cGetTBText (window-handle)

Parameters:

(i) window-handle	Windows handle to a Tool Bar common control.

Returns:

(s) list	List of text associated with each button.

This function retrieves the text for each button of a Tool Bar control. The return string contains each button's text separated by a tab character delimiter. Since button text is "optional", a single tab character designates buttons without text. For example, if there was no text associated with buttons, all that would be returned from this function is a list of tabs.

The class name for the Tool Bar control is "ToolbarWindow32".

Example:

```
;  Add the control manager extender.AddExtender("wwctl34i.dll"); Find the Explorer
tool bar handle
hwndex     = DLLhwnd("Exploring ")
hwnd       = cwndbyclass(hwndex,"Worker")
hwnd       = cwndbyclass(hwnd,"ReBarWindow32")
hwnd       = cwndbyclass(hwnd,"ToolbarWindow32")
buttontext = "Forward"

; Get any button textalltext = cgettbtext(hwnd)

;Get the index of the button
select1=ItemLocate(buttontext, alltext, @tab)
if select1 == 0   Message("Tool Bar Button","%buttontext% Not Found")
   wintext=strreplace(alltext,@tab,@crlf)
   Message("Text found",alltext)
endif
```

See Also:

cClickButton, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetTrackMax

Get the maximum position of a Track Bar control.

Syntax:

cgettrackmax (window-handle)

Parameters:

(i) window-handle	Windows handle to a Track Bar common control.

Return:

(i) Position	Maximum position setting of Track Bar.

This function returns the maximum position of Track Bar control. The maximum position of the control is application specific and can change dynamically.

The class name for the Tool Bar control is "msctls_trackbar32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find the key board track bar handle
Run("rundll32.exe", "shell32.dll,Control_RunDLL main.cpl,Keyboard")
hwndex     = DLLhwnd("Keyboard Properties")
hwnd       = cwndbyclass(hwndex,"#32770")
hwnd       = cwndbyclass(hwnd,"msctls_trackbar32")

; Get the current position.
Position = cgettrackpos(hwnd)

;  Move the position up one
Position = Position + 1

daMax = cGettrackmax(hwnd)
if  Position <= daMax
  if cSettrackpos(hwnd, Position)
    Message("Repeat delay", Position)
  else
    Message("Repeat delay", "%Position% failed")
  endif
else
  Message("Repeat delay", "Currently at max position")
endif
```

See Also:

cGetTrackPos, cGetTrackMin

cGetTrackMin

Get the minimum position of a Track Bar control.

Syntax:

cGetTrackMin (window-handle)

Parameters:

(i) window-handle	Windows handle to a Track Bar common control.

Return:

(i) Postion	Minimum position setting of Track Bar.

This function returns the minimum position of Track Bar control. The minimum position of the control is application specific and can change dynamically.

The class name for the Tool Bar control is "msctls_trackbar32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
; Find the key board track bar handle
Run("rundll32.exe", "shell32.dll,Control_RunDLL main.cpl,Keyboard")

hwndex     = DLLhwnd("Keyboard Properties")
hwnd       = cwndbyclass(hwndex,"#32770")
hwnd       = cwndbyclass(hwnd,"msctls_trackbar32")
; Get the current position.
Position = cgettrackpos(hwnd)
;  Move the position down one
Position = Position - 1
if  position >= cGetTrackmin(hwnd)
   if csettrackpos(hwnd, Position)
      Message("Repeat delay", Position)
   else
      Message("Repeat delay", "%Postition% failed")
   endif
else
  Message("Repeat delay", "Currently at minimum position")
endif
exit
```

See Also:

cGetTrackPos, cGetTrackMax

cGetTrackPos

Get the current position of a Track Bar control.

Syntax:

cGetTrackPos (window-handle)

Parameters:

(i) window-handle	Windows handle to a Track Bar common control.

Return:

(i) Position	Current position of Track Bar.

You can get the current position of a Track Bar control with this function. The class name for the Tool Bar control is "msctls_trackbar32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")
; Find the key board track bar handle
Run("rundll32.exe", "shell32.dll,Control_RunDLL main.cpl,Keyboard")
hwndex     = DLLhwnd("Keyboard Properties")
hwnd       = cwndbyclass(hwndex,"#32770")
hwnd       = cwndbyclass(hwnd,"msctls_trackbar32")
; Get the current position.
Position = cgettrackpos(hwnd)
;  Move the position up one
Position = Position + 1
if csettrackpos(hwnd, Position)
   Message(Repeat delay, Position)
else
   Message(Repeat delay, "%Postition% failed")
endif
```

See Also:

cGetTrackMax, cGetTrackMin

cGetUpDownMax

The function allows you to get the maximum position of an UpDown control.

The class name for this control is "msctls_updown32".

Syntax:

cGetUpDownMax (window-handle)

Parameters:

(i) window-handle	Windows handle to a UpDown common control.

Returns:

(i) Position	Maximum position of the UpDown control.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Internet properties keep history.
Run("rundll32.exe", "shell32.dll,Control_RunDLL inetcpl.cpl,@0")
hwndex   = DLLhwnd("Internet Properties")
hwnd     = cwndbyclass(hwndex,"#32770")
hwnd     = cwndbyclass(hwnd,"msctls_updown32")

; Get the current value.
position = cgetupdownpos(hwnd)

position = position + 1

; In range?
if ( position > cgetupdownmax(hwnd) )
   message( "UpDown", "Position %position% is to large.")
else
   ; Set the up down control.
   csetupdownpos(hwnd, position)
endif
```

See Also:

cSetUpDownPos, cGetUpDownMin, cGetUpDownPos, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetUpDownMin

The function allows you to get the minimum position of an UpDown control.

The class name for this control is "msctls_updown32".

Syntax:

cGetUpDownMin (window-handle)

Parameters:

(i) window-handle	Windows handle to a UpDown common control.

Returns:

(i) Position	Minimum position of the UpDown control.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Internet properties keep history.
Run("rundll32.exe", "shell32.dll,Control_RunDLL inetcpl.cpl,@0")
hwndex   = DLLhwnd("Internet Properties")
hwnd     = cwndbyclass(hwndex,"#32770")
hwnd     = cwndbyclass(hwnd,"msctls_updown32")

; Get the current value.
position = cgetupdownpos(hwnd)

position = position - 1

; In range?
if ( position < cgetupdownmin(hwnd) )
   message( "UpDown", "Position %position% is to small.")
else
   ; Set the up down control.
   csetupdownpos(hwnd, position)
endif
```

See Also:

cSetUpDownPos, cGetUpDownMax, cGetUpDownPos, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetUpDownPos

Get the position of an UpDown control.

Syntax:

cGetUpDownPos (window-handle)

Parameters:

(i) window-handle	Windows handle to a UpDown common control.

Returns:

(i) Position	Current position of the UpDown control.

The function allows you to get the position of an UpDown control. The UpDown control often has a companion control called a buddy window. This function will report an error if the control does not have a buddy window, or if the buddy window has an out-of-range value.

The class name for this control is "msctls_updown32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Internet properties keep history.
Run("rundll32.exe", "shell32.dll,Control_RunDLL inetcpl.cpl,@0")

hwndex   = DLLhwnd("Internet Properties")
hwnd     = cwndbyclass(hwndex,"#32770")
hwnd     = cwndbyclass(hwnd,"msctls_updown32")

; Get the current value.
position = cgetupdownpos(hwnd)

; Adjust the control relative to its current position
position = position + 1

; Set the number of days to keep pages in history via the up down control.
csetupdownpos(hwnd, position)
```

See Also:

cGetUpDownMin, cGetUpDownMax, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cGetWndCursor

Determines an application windows current mouse cursor.

Syntax:

cGetWndCursor (window-handle, request)

Parameters:

(i) window-handle	Windows handle of window of interest.
(i) request	Specifies the operation mode: 1, 2 or 3. (see below)

Returns:

(i) true/false/cursor-id	**@TRUE**, if the window-handle will display a wait cursor and request is set to 3. **@FALSE**, if the window-handle will not display a wait cursor and request is set to 3 **Cursor-id,** The identifier of window-handles current cursor, if request is set to 1 or 2.

You can use this function to determine an application windows current mouse cursor. The mouse cursor is the small image displayed when the mouse pointer is positioned over the window.

cGetWndCursor has three modes of operation. You can indicate your preferred mode by specifying 1, 2 or 3 as the request parameter to the function.

If you pass 1 or 2 as the request parameter to **cGetWndCursor**, the function returns an identifier indicating which system cursor the window will display. Use mode 2 to get the cursor that will display when the mouse pointer is over the center of the target window. If you want to position the mouse pointer over a particular place on the target window and then retrieve the cursor, use mode 1. Mode 1 will get the window center cursor, like mode 2, if the mouse pointer is positioned outside the target windows border.

If you pass in 3 as the second parameter, the function returns **@TRUE** or **@FALSE**. When the displayed cursor is the wait (usually an hourglass) or the starting app (usually an hourglass with an arrow) cursor, **@TRUE** is returned. If the window displays any other cursor type, **@FALSE** is returned.

Any Windows application can change the cursors images. **cGetWndCursor** can recognize changed cursor graphic as long as the target application associates the cursor with one of the Windows cursor identifiers.

The table below shows the Control Manager cursor identifiers associated with each Windows system cursor identifier:

Control Manager Cursor Identifier	Windows System Identifier
1	IDC_ARROW

2	IDC_IBEAM
3	IDC_WAIT
4	IDC_CROSS
5	IDC_UPARROW
6	IDC_SIZE*
7	IDC_ICON*
8	IDC_SIZENWSE
9	IDC_SIZENESW
10	IDC_SIZEWE
11	IDC_SIZENS
12	IDC_SIZEALL
13	IDC_NO
14	IDC_HAND
15	IDC_APPSTARTING
16	IDC_HELP
0	No cursor or unknown cursor

** Obsolete identifiers*

Words of warning: not all applications are well behaved. An application may be busy, i.e. not accepting input, but not display a wait cursor. In this case the function will not help you determine the applications state. Also, select the window handle to pass to the function carefully. In most cases it is best to give **cGetWndCursor** the handle to the target applications main window.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

Cursor0  = "No cursor or Unknown cursor"
Cursor1  = "IDC_ARROW"
Cursor2  = "IDC_IBEAM"
Cursor3  = "IDC_WAIT"
Cursor4  = "IDC_CROSS"
Cursor5  = "IDC_UPARROW"
Cursor6  = "IDC_SIZE"
Cursor7  = "IDC_ICON"
Cursor8  = "IDC_SIZENWSE"
Cursor9  = "IDC_SIZENESW"
Cursor10 = "IDC_SIZEWE"
Cursor11 = "IDC_SIZENS"
Cursor12 = "IDC_SIZEALL"
Cursor13 = "IDC_NO"
Cursor14 = "IDC_HAND"
Cursor15 = "IDC_APPSTARTING"
Cursor16 = "IDC_HELP"

; Start up an application
WindowName = "Netscape"
TestApp = "C:\Program Files\Netscape\Communicator\Program\netscape.exe"
Params =

if !ShellExecute(TestApp, Params, "", @NORMAL, "")
   return
endif

timedelay(2)  ; Need a little time to get a window handle.

; Get a Windows handle
hwnd = DLLhwnd(WindowName)

; Get the current cursor
```

```
AppCursorId = cgetwndcursor(hwnd, 2)

; Hourglass or arrow-hourglass?
If cgetwndcursor(hwnd, 3)
   MessageTitle = "%WindowName% is busy"
else
   MessageTitle = "%WindowName% is not busy"
endif

WndCursor   = Cursor%AppCursorId%
message(MessageTitle, "Its cursor is %WndCursor%.")
winclose(WindowName)
```

See Also:

cWndInfo, MouseMove (WIL Reference Manual)

cPostButton

Clicks a button, emulating a user. This uses an undocumented method, different from cClickButton, that seems to work when cClickButton causes the script to hang.

Syntax:

cPostButton(hwnd)

Parameters:

(i) hwnd	PushButton handle from a cWndBy... function.

Returns:

(0)	always 0.

This function is designed to click a push button. As opposed to cClickButton, this method may be tried when the cClickButton appears to cause the script to hang. If your script seems to hang after a cClickButton call, try the cPostButton function instead.

Example:

```
AddExtender("wwctl34i.dll")
Run("notepad.exe","")
SendMenusTo("~Notepad","FileOpen")
WinWaitExist("Open",5)
openwnd=DllHwnd("Open")
editwnd=cWndByID(openwnd,1152)
mask=cSetEditText(editwnd, "*.*")
openbutton=cWndByName(openwnd,"Open")
cPostButton(openbutton)
Message("Open file mask changed to *.*","and Open button clicked.")
exit
```

See Also:

cClick Button , cCheckBox, cRadioButton

cPostMessage

The real PostMessage (or PostMessageA if you want to get nitpicky).

Syntax:

cPostMessage(hWnd, msg, wparam, lparam)

Parameters:

(i) hWnd	Window handle
(i) msg	Message number
(i) wparam	First message parameter
(i) lparam	Second message parameter (must be numeric)

Returns:

(i)	The value returned by the Windows "PostMessage" function.

This function is designed to allow sophisticated users to access the Windows API "PostMessage" function directly. **cPostMessage** calls the Windows API "PostMessage" function with the specified parameters and sends the specified message to the given window.

Example:

```
AddExtender("wwctl34i.dll")
Run("notepad.exe","")
SendMenusTo("~Notepad","FileOpen")
WinWaitExist("Open",5)
openwnd=DllHwnd("Open")
editwnd=cWndByID(openwnd,1152)
mask=cSetEditText(editwnd, "*.*")
openbutton=cWndByName(openwnd,"Open")
;cClickButton(openbutton)   ;We will emulate a cClickButton using
; a PostMessage instead of the built in SendMessage the
; function actually uses
cPostMessage(openbutton,245,0,0)   ; 245 == BM_CLICKED
Message("Open file mask changed to *.*","and Open button clicked.")
exit
```

See Also:

IntControl 22 (WIL Reference Manual)

cRadioButton

Examines or sets a radio button.

Syntax:

cRadioButton(hwnd, flag)

Parameters:

(i) hwnd	RadioButton handle from a cWndBy... function.
(i) flag	See below.

Returns:

(i)	Previous state of checkbox.

Flags

-1	Examine the settings of a RadioButton
	0 Not Checked
	1 Checked
1	Sets a RadioButton

This function is used to examine or modify the status of radio buttons. Note that radio buttons cannot be cleared. To clear a radio button, another radio button must be set..

Example:

```
;Note:  This example designed for Windows 95

;Further note that this example shows how to navigate thru
;a tabbed dialog

AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL mmsys.cpl")

title="Multimedia Properties"
multiwnd=DllHwnd(title)
tabtitle="Video"

systabwnd=cWndById(multiwnd,12320)
cSetTabItem(systabwnd,2)

; confirm correct tab
videohwnd=cWndBySeq(multiwnd,1)
thistitle=cWndInfo(videohwnd,0)
if thistitle!=tabtitle
   Message("Error","Incorrect tab title")
   exit
endif

fullscreenradio=cWndByName(videohwnd,"Window")
rad=cRadioButton(fullscreenradio,-1)
Message("Video: Window Radio button set to",rad)
exit
```

See Also:

cCheckBox, cClickButton

cSendMessage

The real SendMessage (or SendMessageA if you want to get nitpicky).

Syntax:

cSendMessage(hWnd, msg, wparam, lparam)

Parameters:

(i) hWnd	Window handle
(i) msg	Message number
(i) wparam	First message parameter
(i) lparam	Second message parameter (must be numeric)

Returns:

(i)	The value returned by the Windows "SendMessage" function.

This function is designed to allow sophisticated users to access the Windows API "SendMessage" function directly. **cSendMessage** calls the Windows API "SendMessage" function with the specified parameters and sends the specified message to the given window.

Example:

```
WMCLOSE = 16                ; value of WM_CLOSE message
AddExtender("wwctl34i.dll")
hwnd = DllHwnd("~Notepad")
If hwnd != 0
    cSendMessage(hwnd, WMCLOSE, 0, 0)  ; send message
Endif
```

See Also:

IntControl 22 (WIL Reference Manual)

cSetCalDate

Set the date or date range of a Month Calendar control with this function.

Syntax:

cSetCalDate (window-handle, date-time, date-time)

Parameters:

(i) window-handle	Windows handle to a Month Calendar common control.
(s) date-time	Time in YYYY:MM:DD:HH:MM:SS format
(s) date-time	Time in YYYY:MM:DD:HH:MM:SS format

Return:

(i)	**@TRUE**, if new date-time set **@FALSE**, if new date-time was not set

Set the date or date range of a Month Calendar control with this function. Month Calendar controls may or may not allow a date range to be specified. If the control you are setting allows the selection of a range of dates, you must place a valid date in both the second and third parameters. You can select a single date in a range style control by placing the same date in both parameters. If the control accepts only single dates, not ranges, the function will simply ignore the content of the third parameter. The Month Calendar data control does not display the time of day, so the hour, minute and second can be omitted from the input parameters. The colon delimiters must be provided, however.

The class name for the Month Calendar control is "SysMonthCal32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Get a handle
hwndex = DLLhwnd("Microsoft Control Spy")
hwnd   = cwndbyclass(hwndex,"CSPYContainer")
hwnd   = cwndbyclass(hwnd,"SysMonthCal32")

startdate = "1997:12:31:::"
enddate   = "1998:1:2:::"
if csetcaldate(hwnd, startdate, enddate)
   message("Set Calendar", "Start date = %startdate%  Enddate = %enddate%")
else
   message("Set Calendar", "Failed")
endif
```

See Also:

cGetCalDate, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cSetCBItem

Sets a single item in a standard Combobox.

Syntax:

cSetCBItem(hwnd, item)

Parameters:

(i) hwnd	Combobox handle from a cWndBy... function.
(i) item	Item number to set (first one is 1).

Returns:

(i)	1 if successful, 0 on failure

This function is designed to set a single item in a standard Combobox.

Example:

```
;Note:  This example designed for Windows 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

netwnd=DllHwnd("Network")
cfghwnd= cWndByName(netwnd,"Configuration")
cbWnd=cWndbyID(cfghwnd,405)
response=cGetCBText(cbwnd)
logon=ItemLocate("Windows Logon", response, @tab)

if logon==0
   Message("Windows Logon","Not Found")
else
   cSetCBItem(cbwnd,logon)
   Message("Windows Logon","Selected")
endif
exit
```

See Also:

cSetLBItem, cGetCBText

cSetDTPDate

Change the values of a Date Time Picker control.

Syntax:

cSetDTPDate (window-handle, date-time)

Parameters:

(i) window-handle	Windows handle to a Date Time Picker common control.
(s) date-time	Time in YYYY:MM:DD:HH:MM:SS format.

Returns:

(i)	**@TRUE**, if new date-time set **@FALSE**, if new date-time was not set

You can change the values of a Date Time Picker control with this function. The input date and time must be in the standard datetime format. The function accepts both two and four digit years. It assumes that years 50-99 are from 1950 to 1999 and years 00-49 are years 2000 to 2049. If you wish to input years from the first century, prefix the years with two zeros, i.e., 0049.

A control may not display all parts of the standard date-time format. However, an error will occur if any part is missing from the date-time string. This is because the control stores all parts of the format even if it only displays some parts. A Date Time Picker control may have a minimum and maximum range. This function will report an error if an input date falls outside of this range.

The class name for this control is "SysDateTimePick32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Get a handle
hwndex = DLLhwnd("Microsoft Control Spy")
hwnd   = cwndbyclass(hwndex,"CSPYContainer")
hwnd   = cwndbyclass(hwnd,"SysDateTimePick32")

startdate = "1899:2:28:0:0:0"
if csetdtpdate(hwnd, startdate)
   message("Set date/time", startdate)
else
   message("Set date/time", "Failed")
endif
```

See Also:

cWndByClass, cWndinfo, DLLhwnd (WIL help).

cSetEditText

Changes the contents of an Edit Box.

Syntax:

cSetEditText(hwnd, newtext)

Parameters:

(i) hwnd	EditBox handle from a cWndBy... function.
(s) newtext	New EditBox contents.

Returns:

(i)	1 if successful, 0 if operation failed.

This function is used to modify the contents of edit controls.

Example:

```
AddExtender("wwctl34i.dll")
Run("notepad.exe","")
SendMenusTo("~Notepad","FileOpen")
WinWaitExist("Open",5)
openwnd=DllHwnd("Open")
editwnd=cWndByID(openwnd,1152)
mask=cSetEditText(editwnd, "*.*")
Message("Open file mask changed to","*.*")
exit
```

See Also:

cGetEditText, cSetLBItem, cSetCBItem

cSetFocus

This function sets the keyboard focus to the control window specified by the input window handle.

Syntax:

cSetFocus(window-handle)

Parameters:

(i) window-handle	Windows handle to control.

Returns:

(i)	**@TRUE**, Keyboard focus set to window **@FALSE**, Unable to set keyboard focus.

When a window has the keyboard focus, keystrokes are directed to the window. The function will return **@FALSE**, if it is unable to activate the parent window of the targeted control window. Call the **WinActivate** function with the name of the controls parent window before calling **cSetFocus**, if this problem occurs.

It is best to perform keyboard tasks immediately after calling this function. This is because keyboard focus is a temporary window state and system activity can quickly switch it to another window.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Start Notepad
Run("notepad.exe","")

; Activate the Print Setup dialog.
SendMenusTo("~Notepad","Filepage setup")
WinWaitExist("Page Setup",5)

; Get a windows handle to the dialog.
openwnd=DllHwnd("Page Setup")

; Get a windows handle to the "Footer" edit control (control id = 31).
editwnd=cWndByID(openwnd,31)

;  Set the keyboard focus to the footer control.
cSetFocus(editwnd)

; Select the existing text and type in new footer text.
sendkey("+{right 20}")
sendkey("Figure &p")
```

See Also:

cWndByID, cGetEditText, cSetEditText, WinActivate,DLLhwnd

cSetIpAddr

This function places an IP address stored in an IP Address common control.

The class name for this control is "SysIPAddress32".

Syntax:

cSetIpAddr(window-handle, IP address)

Parameters:

(i) window-handle	Window handle to an IP Address Controls.
(s) IP Address	Text string containing Internet protocol (IP) address.

Returns:

(i)	**@TRUE**, if address has been placed in control **@FALSE**, if address could not be set.

Note: Several Windows system property sheets contain controls that look and behave like an IP address control. However, these controls are not a common control and have different class names. This function does make an attempt to identify and set an IP address for these pseudo common controls.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find Control Spy ip address window handle.
;hwnd = DLLhwnd("Microsoft Control Spy -")
;hwnd = cwndbyclass(hwnd, "CSPYContainer")
;hwnd = cwndbyclass(hwnd,"SysIPAddress32")

; Get the address.
if csetipaddr(hwnd,"254.254.254.254")

; Test ip address get function
   ipaddr = cgetipaddr(hwnd)

; Get the window class name and display the new IP address.
wintext = cwndinfo(hwnd, "2")
if wintext == ""
          wintext = "a Classless Window"
endif
message("IP Address from %wintext%.", "IP Address: %ipaddr%.")
endif
```

See Also:

cGetIpAddr, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cSetLBItem

Sets a single item in a standard single item list box.

Syntax:

cSetLBItem(hwnd, item)

Parameters:

(i) hwnd	Listbox handle from a cWndBy... function.
(i) item	Item number to set (first one is 1).

Returns:

(i)	1 if successful, 0 on failure.

This function is designed to set a single item in a standard single item list box. It will not work with a multiple selection listbox.

Example:

```
;Note:  This example designed for Windows 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

netwnd=DllHwnd("Network")
cfghwnd= cWndByName(netwnd,"Configuration")
lbWnd=cWndbyID(cfghwnd,404)
response=cGetLBText(lbwnd)
dialup=ItemLocate("TCP/IP -> Dial-Up Adapter", response, @tab)
if dialup==0
   Message("TCP/IP Dialup Adapter","Not Found")
else
   cSetLBItem(lbwnd,dialup)
   Message("TCP/IP Dialup Adpater","Selected")
endif
exit
```

See Also:

cSetCBItem, cGetLBText

cSetLBItemEX

Select one or more items in a List Box control.

Syntax:

cSetLBItemEX (window-handle, Item-#-list)

Parameters:

(i) window-handle	Windows handle to a List Box control.
(s) Item # list	Numbers of List Box items to select.

Returns:

(i)	@TRUE, If all listed items were selected. @FALSE, If one or more listed items were not selected.

This function allows you to select one or more items in a List Box control. It extends the Control Manager function cSetLBItem by allowing interactions with multiple-select list boxes. Multiple-select List Boxes allow the user to highlight more than one item in the list box at a time. You specify the items to select by listing one or more space separated numbers as a string in the second parameter. Each number should correspond to an item in the list starting with 1 as the item at the top of the list.

The function returns an error message, if you pass in more than one item number but the specified List Box does not support multiple-selection. It will also return an error message, if you specify any item numbers greater than the number of items in the list.

The class name for List Box controls is "List Box.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

;Get the handle of the parent window of
;the list box control.

hwnd = cwndbyclass(hwndParent, "ListBox")

; Get all the item text for the list box.
AllItems = cGetLBText(hwnd)

; Get a count of the list choices
ItemCnt = ItemCount(AllItems, @TAB)

;Create a space delimited list of item numbers
NumberList = ""
for i = 1 to ItemCnt
   NumberList = ItemInsert(i, -1, NumberList, " ")
next

; Select all the items in the list
if ! cSetLBItemEX(hwnd, NumberList)
   goto ErrorHandler
endif
```

See Also:

cGetLBText, cClearLBAll, cGetLBSelText, cSetLBItem

cSetLVItem

You can use this function to make an item of a List View common control the selected item. Once an item is selected, it is the ultimate recipient of mouse clicks and key strokes sent to the control.

The class name for this control is "SysListView32".

Syntax:

cSetLVItem (window-handle, item)

Parameters:

(i) window-handle Windows handle to a List View common control.

(i) item The one based number of the item to make the selected item, i.e., the item's location in a list whose first item is item #1.

Returns:

(i) **@TRUE**, if is selected.
@FALSE, if item was not selected.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Explorer List View area
hwndex = DLLhwnd("Exploring ")
hwnd   = cwndbyclass(hwndex,"SHELLDLL_DefView")
hwnd   = cwndbyclass(hwnd,"SysListView32")
myitem = "Backup"

; Get the labels from the first column of each row in the control.
wintext =  cgetlvtext(hwnd)

;Get the index of the item
select1=ItemLocate(myitem, wintext, @tab)

if select1

   ; Select the item from the list control.
   if csetlvitem(hwnd, select1)
          cdblclickitem(hwnd, "")
   else
          ; Can't select the item
          message("Item not selected",  "Item number %select1% may no longer
exist.")
   endif
else
   message("Item not found",  "Item number %select1% may no longer exist.")
endif
```

See Also:

cGetLVText, cClearLVItem, cDblClickItem, cWndByClass, cWndinfo, DLLhwnd (WIL help).

cSetTABItem

Selects a tab from a tabbed dialog.

Syntax:

cSetTABItem(hwnd, item)

Parameters:

(i) hwnd	SysTabControl32 handle from a cWndBy... function.
(i) item	Item number to set (first one is 1), i.e., the item's location in a list whose first item is item #1.

Returns:

(i)	1 if successful, 0 on failure

This function is designed select a tab from a tabbed dialog.

Example:

```
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL mmsys.cpl")
title="Multimedia Properties"
multiwnd=DllHwnd(title)
tabtitle="Video"
systabwnd=cWndByClass(multiwnd,"SysTabControl32")
cSetTabItem(systabwnd,2)

; confirm correct tab
videohwnd=cWndBySeq(multiwnd,1)
thistitle=cWndInfo(videohwnd,0)
if thistitle!=tabtitle
   Message("Error","Incorrect tab title")
   exit
endif
```

See Also:

cRadioButton, cCheckBox, cSetEditText

cSetTrackPos

Set the position of a Track Bar control.

Syntax:

cSetTrackPos (window-handle, position)

Parameter:

(i) window-handle	Windows handle to a Track Bar common control.
(i) position	New position of Track Bar.

Return:

(i)	**@TRUE**, if new position set. **@FALSE**, if position could not be set.

Use this function to set the position of a Track Bar control. Track Bars have an associated position range that can change dynamically. This function returns an error message if the position parameter is outside this range.

The class name for the Tool Bar control is "msctls_trackbar32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find the key board track bar handle
Run("rundll32.exe", "shell32.dll,Control_RunDLL main.cpl,Keyboard")
hwndex     = DLLhwnd("Keyboard Properties")
hwnd       = cwndbyclass(hwndex,"#32770")
hwnd       = cwndbyclass(hwnd,"msctls_trackbar32")

;  Set position.
Position = 1

if  Position > cgettrackmax(hwnd)
   message("Track Bar", "position is to large.")
else
  if Position < cgettrackmin(hwnd)
      message("Track Bar", "position is to small.")
  else
      if csettrackpos(hwnd, Position)
         Message("Track Bar", "Set position to %Position%")
      else
         Message("Track Bar", "Failed  to set position %Position%")
      endif
  endif
endif
```

See Also:

cGetTrackPos, cGetTrackMax, cGetTrackMin

cSetTVItem

This function is used to select an item of a Tree View common control. Once an item is selected, it is the ultimate recipient of mouse clicks and keystrokes sent to the control. Placing the text of an item and each of its ancestor items starting at the root into a string specifies an item. A tab character must delimit each items text.

The class name for this control is "SysTreeView32".

Syntax:

cSetTVItem (window-handle, item-path)

Parameters:

(i) window-handle	Windows handle to a Tree View common control.
(s) item-path	Tab-delimited list of item text.

Returns:

(i)	**@TRUE**, if item is selected. **@FALSE**, if item is not selected.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find the Explorer tree control
hwnd      = DLLhwnd("Exploring")
hwnd      = cwndbyclass(hwnd, "BaseBar")
hwnd      = cwndbyclass(hwnd, "ReBarWindow32")
hwnd      = cwndbyclass(hwnd,"SysTreeView32")

; Path to the ctlmgr item (note the space before (C)).
selection = "Desktop%@TAB%My Computer%@TAB% (C:)%@TAB%ctlmgr"

; Select the item from the tree control.
if csettvitem(hwnd, selection)

   ; Double click the selected item.
   if !cdblclickitem(hwnd, "")
         Message ("cddblckikcitem", "Failed")
   endif
else
   ; Can't select the item
   selection = strreplace(selection,@TAB, @CRLF)
   message("Item not selected",  "Node tree:%@CRLF% %selection% %@CRLF%- not
found.")

endif
```

See Also:

cClearTVItem, cDblClickItem, cWndByClass, cWndinfo, DLLhwnd (WIL help).

cSetUpDownPos

The function can be used to set the position of an UpDown control.

Syntax:

cSetUpDownPos (window-handle, position)

Parameters:

(i) window-handle	Windows handle to a List View common control.
(i) position	New position for the Updown control.

Returns:

(i)	**@TRUE**, if position is changed. **@FALSE**, if position is not change.

The UpDown control often has a companion control called a buddy window. For example, an UpDown control with an edit box for a buddy window becomes a spinner control. Changing the position of the UpDown control changes the content of the buddy window. UpDown controls can have a minimum and maximum range, and this function will generate an error if the input value is out of range.

The class name for this control is "msctls_updown32".

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Internet properties keep history.
Run("rundll32.exe", "shell32.dll,Control_RunDLL inetcpl.cpl,@0")
hwndex   = DLLhwnd("Internet Properties")
hwnd     = cwndbyclass(hwndex,"#32770")
hwnd     = cwndbyclass(hwnd,"msctls_updown32")

; Get the current value.
position = cgetupdownpos(hwnd)

position = position + 1

; Set the number of days to keep pages in history via
; the up down control.
csetupdownpos(hwnd, position)
```

See Also:

cGetUpDownPos, cWndByClass, cWndInfo, DLLhwnd (WIL help).

cSetWndText

Change the text associated with a Windows control.

Syntax:

cSetWndText(window-handle, text)

Parameters:

(i) window-handle	*Window handle* of a control.
(s) text	New text to be associated with control

Returns:

(i) **@FALSE**	Always returns 0

You can use **cSetWndText** to change the text associated with a Windows control. The text changed by this function depends on the type of control you target. For windows with a title bar, the text in the title bar is changed to the new string. On the other hand, *Edit* controls have the text in the edit box changed, unless they have the *WS_CAPTION* style, then the caption is changed. *Combo boxes* have the text in the edit control part changed. *Button* controls have the text displayed on the button changed.

cSetWndText will error, if the input *window handle* does not reference an existing window or control.

When **cSetWndText** is applied to a *top-level* (parent) window, it is similar to the *WIL* function **WinTitle**. See Windows Interface Language documentation for more information on the **WinTitle** function.

Warning: Some applications may rely upon their windows and controls title staying the same! Therefore, like all control manager functions, the **cSetWndText** function should be used with caution.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Launch the save as dialog (assumes Notepad is running)
SendMenusTo("~Notepad", "File Open")
WinWaitExist("Open", 8)

; Get a button control's handle
hwnd      = DLLhwnd("Open")
hwndCtl   = cWndByName(hwnd, "Cancel")

; Change the buttons text.
sChange = "Quit"
cSetWndText(hwndCtl, sChange)

; Make sure it actually changed.
sNew  =   cWndinfo(hwndCtl, 0)
if strcmp(sNew, sChange) != 0
   goto errorhandler
endif
```

See Also:

cWndinfo

cWinIdConvert

Convert between a Windows handle and a Window Id (pseudo-handle)

Syntax:

cWinIdConvert(window-id | window-handle)

Parameters:

(s/i) window-id or window-handle	Window ID handle **or** Window handle.

Returns:

(s/i) window-id or window-handle	Window handle, if parameter is a Window ID **or** Window ID, if parameter is a window handle.

You can use **CWinIdConvert** to convert between a Windows handle and a Window Id (pseudo-handle). If you pass in a Windows handle, the function returns the corresponding Window ID. Conversely, if you pass in a Window ID, the function returns the corresponding Window handle.

CWinIdConvert will error if the input value does not reference an existing window.

Example:

```
;** Convert from win-ID to win-handle **
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Get notepads Edit control.
Run("notepad.exe", "My.txt")
WinId = WinIdGet("~Notepad")
hWnd = cWinIdConvert(WinId)
hEdit = cWndByClass(hWnd, "Edit")

; Get the text from NotePad.
sMyText = cgetedittext(hEdit)

; Start another notepad.
Run("notepad.exe", "")
WinId = WinIdGet("~Notepad")
hWnd = cWinIdConvert(WinId)
hEdit = cWndByClass(hWnd, "Edit")

; Move the text to the second NotePad
cSetEditText(hEdit, sMyText)
Exit

;** Convert from win-handle to win-ID **
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Get notepad's Edit control.
Run("notepad.exe", "My.txt")
WinId = WinIdGet("~Notepad")
```

```
hWnd = cWinIdConvert(WinId)
hEdit = cWndByClass(hWnd, "Edit")

; Get the text from NotePad.
sMyText = cgetedittext(hEdit)

; Start another notepad.
Run("notepad.exe", "")
WinId = WinIdGet("~Notepad")
hWnd = cWinIdConvert(WinId)
hEdit = cWndByClass(hWnd, "Edit")

; Move the text to the second NotePad
cSetEditText(hEdit, sMyText)

; Launch toe Save file dialog.
SendMenusTo(WinId, "File Save As")
WinWaitExist("Save As", 8)

; Get file name edit window id.
hWndDialog = cWndByName(0,"Save As")
hEdit      = cWndByClass(hWndDialog, "Edit")
EditId     = cWinIdConvert(hEdit)

; Give the file a name and press the save button.
SendKeysTo(EditId, "New File.txt")
hButton    = cWndByName(hWndDialog, "Save~")
cPostButton( hButton )
```

See also:

WinIdGet, DllHwnd, WinItemNameId (See WIL help file for documentation of these functions)

cWndByClass

Obtains the handle of a child window or control based on the window or control class.

Syntax:

cWndByClass(hWnd, "ControlClass")

Parameters:

(i) hWnd	Handle of parent window.
(s) ControlClass	Class of window ot control.

Returns:

(i)	Handle to the child window or control.

This function obtains the handle of a child window or control for use in subsequent operations. The first parameter, hWnd, is a handle obtained via one of these functions:

DllHwnd (See Main WIL documentation)
cWndByClass
cWndByID
cWndByName
cWndBySeq
cWndInfo (some parameter values)

Note: cWndByClass accepts 0 or "" as the first parameter, which is the handle of parent window. It will search all top-level windows for the second parameter instead of searching the child windows of parameter 1.

Example:

```
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL mmsys.cpl")
title="Multimedia Properties"
multiwnd=DllHwnd(title)
tabtitle="Video"
systabwnd=cWndByClass(multiwnd,"SysTabControl32")
cSetTabItem(systabwnd,2)

; confirm correct tab
videohwnd=cWndBySeq(multiwnd,1)
thistitle=cWndInfo(videohwnd,0)
if thistitle!=tabtitle
   Message("Error","Incorrect tab title")
   exit
endif
```

See Also:

cWndByName, cWndBySeq, cWndById, cWndInfo

cWndByID

Obtains the handle of a child window or control based on the window or control ID number.

Syntax:

cWndByID(hWnd, ID-number)

Parameters:

(i) hWnd	Handle of parent window.
(i) ID-number	ID number of control.

Returns:

(i)	Handle to the child window or control.

This function obtains the handle of a child window or control for use in subsequent operations. The first parameter, hWnd, is a handle obtained via one of these functions:

DllHwnd (See Main WIL documentation)
cWndByClass
cWndByID
cWndByName
cWndBySeq
cWndInfo (some parameter values)

Note: cWndByID accepts 0 or "" as the first parameter, which is the handle of parent window. It will search all top-level windows for the second parameter instead of searching the child windows of parameter 1.

Example:

```
;Note:  This example designed for Window 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

netwnd=DllHwnd("Network")
cfghwnd= cWndByName(netwnd,"Configuration")
lbWnd=cWndbyID(cfghwnd,404)
response=cGetLBText(lbwnd)
response=StrReplace(response,@tab,@crlf)
Message("Response",response)
exit
```

See Also:

cWndByName, cWndBySeq, cWndInfo

cWndByName

Obtains the handle of a child window or control based on the window or control title.

Syntax:

cWndByName(hWnd, "ControlTitle")

Parameters:

(i) hWnd	Handle of parent window.
(s) ControlTitle	Title of window or control (without any &'s). Accepts a partial-window-name.

Returns:

(i)	Handle to the child window or control.

This function obtains the handle of a child window or control for use in subsequent operations. The first parameter, hWnd, is a handle obtained via one of these functions:

DllHwnd (See Main WIL documentation)
cWndByClass
cWndByID
cWndByName
cWndBySeq
cWndInfo (some parameter values)

Note: **cWndByName** accepts 0 or "" as the first parameter, which is the handle of parent window. It will search all top-level windows for the second parameter instead of searching the child windows of parameter 1.

See Partial Window Names for a detailed explanation of the rules governing, partial window names.

Example:

```
;Note:  This example designed for Window 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

netwnd=DllHwnd("Network")
cfghwnd= cWndByName(netwnd,"Configuration")
lbWnd=cWndByID(cfghwnd,404)
response=cGetLBText(lbwnd)
response=StrReplace(response,@tab,@crlf)
Message("Response",response)
exit
```

See Also:

cWndBySeq, cWndById, cWndInfo

cWndBySeq

Obtains the handle of a child window or control based on the window or control sequence number.

Syntax:

cWndBySeq(hWnd, seq-number)

Parameters:

(i) hWnd	Handle of parent window.
(i) seq-number	Sequence number of control (starting at 1).

Returns:

(i)	Handle to the child window or control.

This function obtains the handle of a child window or control for use in subsequent operations. The first parameter, HWnd, is a handle obtained via one of these functions:

DllHwnd (See Main WIL documentation)
cWndByClass
cWndByID
cWndByName
cWndBySeq
cWndInfo (some parameter values)

Note: cWndBySeq accepts 0 or "", as the first parameter, which is the handle of parent window. It will search all top-level windows for the second parameter instead of searching the child windows of parameter 1.

Example:

```
;Note:  This example designed for Window 95
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

netwnd=DllHwnd("Network")
cfghwnd= cWndByName(netwnd,"Configuration")
lbWnd=cWndbySeq(cfghwnd,2)
response=cGetLBText(lbwnd)
response=StrReplace(response,@tab,@crlf)
Message("Response",response)
exit
```

See Also:

cWndByName, cWndById, cWndInfo

cWndByWndSpec

Returns a window handle based on the window specifications.

Syntax:

cWndByWndSpec(window-class, window-module, child-count [, child-id...])

Parameters:

(s) window-class	Window class name
(s) window-module	Window module name.
(i) child-count	Number of child windows (must be at least 0 and no greater than 13)
(i) child-id	Child window id (minimum 0, maximum 13)

Returns:

(i) window-handle	Window handle of specified window.

Place the components of the window specification in the parameters to the function. The first parameter must contain the window's class name.

The second contains the module name. The module name is the name of the executable or DLL file that is responsible for the window. It does not include the file name's path or extension.

The third paramter must contain the number of child window ids given in the remaining parameters. These three parameters are always required. The remaining parameters consist of child window ids: one child window id per parameter. The number of ids must correspond to the number given in the child-count or third parameter. If the number of child id parameters does not match the value in the child-count parameter your will get an "Invalid window specification" error. There is a limit of 13 child window parameters. You can use this function on windows with more than 13 child windows but only thirteen child ids can be used to identify the window.

The function returns zero (0), if the specified window is not found.

You can create your own window specification or have Control Manager create one for you by calling the **cWndGetWndSpec** function, which is recommended. To obtain a window specification, write a WIL script that simply calls **cWndGetWndSpec** with the target window's handle and save the result. The specification can be saved to the clipboard, to a file or displayed in a message box and hand copied.

Although a window specification eliminates a lot of the uncertainty in obtaining a window's handle, there is still some chance for error. For example, if you have two instances of the same application running, they will have the same windows specification in most cases. The function resolves ambiguities like this, by selecting the specification-matching window that is closest to the top in Z order. The Z order of a window indicates the window's position in a stack of overlapping windows on the Windows desktop. The window at the top of the Z order overlaps all other windows. All other windows overlap the window at the bottom of the Z order.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Look for Win2k's Notepad by its specification.
hWnd = cWndByWndSpec("Notepad", "notepad", 1, 15)

; Success?
if hWnd != 0 then message("Found NotePad","")
else message("Notepad not found","")
```

See Also:

cWndGetWndSpec, cWndByClass, cWndByName, cFindbyClass, cFindByName

cWndExist

Determines if a child window of the specified parent exists.

Syntax:

cWndExist (window-handle, window-info, info-type)

Parameters:

(i) window-handle	parent windows handle.
(i/s) child-attribute	either window name, window sequence, window id or window class
(i) attribute-type	1 = window name, 2 = window sequence, 3 = window id 4 = window class

Returns:

(i) Window-handle	Windows handle if found, otherwise, 0

Use this function to determine if a child window of the specified parent exists. It can also be used to obtain the Windows handle that can be passed to other Control Manager functions. The function searches for a child window based on the window attribute in the second parameter. You need to tell the function about the content of the attribute by placing a number from 1 to 4 in the third parameter (see the table below).

The return value is the Windows handle to the child window with the specified attribute. If no child with the attribute can be found, the function will return 0.

This function is a combination of the four Control Manager "By" functions; cWndByName, cWndBySeq, cWndById and cWndByClass. The main difference is that this function returns immediately, if the window with the indicated attribute does not exist. The four "By" functions will make multiple attempts at finding a window. These attempts can delay a script for up to 8 seconds when the child window does not exist.

If the child attribute specified is a window-name, See Partial Window Names for a detailed explanation of the rules governing, partial window namesl.

The number in the third parameter has the following meanings:

Value	Description	Function
1	The text is the child window caption.	cWndByName
2	The number is the sequence number of the child window.	cWndBySeq
3	The number is the Id number of the child window.	cWndById
4	The text is the class name of the child	cWndByClass

window.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Give some symbolic names to the "by" method parameter.
ByName  = 1
ByClass = 4
ById    = 3
BySeq   = 2

appName = "YATS32"
hwnd    = DLLhwnd(appName)

; Test by class.
ClassName =  "ListBox"
hwnd1 = cWndExist(hwnd, ClassName, ByClass)
hwnd2 = cwndbyclass(hwnd, ClassName)
if hwnd1 != hwnd2
   goto SlowWindowFound
endif

; Test by name.
WindowName = "Time Servers"
hwnd1= cwndbyname(hwnd, WindowName)
hwnd2= cWndExist(hwnd, WindowName, ByName)
if hwnd1 != hwnd2
   goto SlowWindowFound
endif

; Test by Control Id.
WindowId = 65535
hwnd1 = cWndExist(hwnd, WindowId , ById)
hwnd2 = cwndbyId(hwnd,WindowId )
if hwnd1 != hwnd2
   goto SlowWindowFound
endif

; Test by sequence.
WindowSequence = 1
hwnd1 = cWndExist(hwnd, WindowSequence, BySeq)
hwnd2 = cwndbySeq(hwnd, WindowSequence )
if hwnd1 != hwnd2
   goto SlowWindowFound
endif
```

See Also:

cWndById, cWndByName, cWndBySeq, cWndByClass, DLLhwnd

cWndGetWndSpec

Creates a window specification from the window handle.

Syntax:

cWndGetWndSpec(window-handle)

Parameters:

(i) window-handle	Window handle of top-level window to create specification for.

Returns:

(s) window-specification	Specification of indicated window.

This function is a script development tool that creates a window specification from the window handle you supply as the only parameter. You can save window specifications in WIL scripts as the parameters to the **cWndByWndSpec** function. This allows you to more accurately obtain an application's top-level window handle than by using the DLLhwnd function with a window title. The window specification is a more precise window identification method than the window title approach, because applications often change their titles on the fly and partial window titles can identify more than one application's top-level window.

The function's parameter must be the window handle to an existing top-level window. A top-level window is any window that does not have the WS_CHILD style and is usually the main window of an application. The function will generate an error if the passed in window handle is not to a top-level window.

The window specification consists (in order) of the window's class name, module name, number of child windows and child window ids. Specification elements are separated by comas (,). The module name is the name of the executable or DLL file that is responsible for the window. It does not include the file name's path or extension. Both the class name and module name are surrounded by double quotes ("). There is a limit of 13 child windows allowed. You can obtain the window specification of a window with more than 13 child windows but only the first thirteen will be included in the specification.

Example:

```
;  Add the control manager extender.
AddExtender("wwctl34i.dll")

; Find Notepad's Windows handle
hwnd = DLLhwnd("~Notepad")

; Get the window specification.
Spec = cWndGetWndSpec(hwnd)

; Get the handle by the window specification.
hWndSpec = cWndByWndSpec(%Spec%)

; Make sure the result matches the original handle
if hwnd == hWndSpec then message("Window Spec is correct", Spec)
else message("Window Spec is NOT correct", Spec)
```

See Also:

cWndByWndSpec, cWndByName

cWndInfo

This function returns information about a window or control.

Syntax:

cWndInfo(hWnd, request)

Parameters:

(i) hWnd	Window or control handle.
(i) request	Request code for desired information (see below).

Returns:

(s)	Desired Information.

Request codes

0	Title
1	ID
2	Class
3	Parent
4	First Sibling
5	Previous Sibling
6	Next Sibling
7	Last Sibling
8	First Child
9	Owner
20	Window style
21	Window extended style
22	Class style

Example:

```
;Note:  This example designed for Windows 95
;
;Further note that this example shows how to navigate thru
;a tabbed dialog
;
AddExtender("wwctl34i.dll")
Run("rundll32.exe", "shell32.dll,Control_RunDLL netcpl.cpl")

title="Network"
titlewnd=DllHwnd("Network")
tabtitle="Identification"

while 1
   thishwnd=cWndInfo(titlewnd,8)
   thistext=cWndInfo(thishwnd,0)
   if thistext==tabtitle then break
   SendKeysTo(title,"^{pgdn}")
   TimeDelay(0.5)
```

```
end while

Message(tabtitle,"Activated")
exit
```

See Also:

cWndByName, cWndByClass, cWndById, cWndBySeq

cWndState

Get the current state of a window or control.

Syntax:

cWndState(window-handle / control-handle)

Parameters:

(i) hWnd	handle to window or control .

Returns:

(i)	Desired Information. See Below.

Possible return values for a window:

Constant	Value	Description
@HIDDEN	**-1**	Specified window exists, but is hidden.
@FALSE	**0**	Specified window does not exist.
@ICON	**1**	Specified window is iconic (minimized).
@NORMAL	**2**	Specified window is a normal window.
@ZOOMED	**3**	Specified window is zoomed (maximized).

Possible return values for a control:

Constant	Value	Description
@HIDDEN	**-1**	Specified control exists, but is hidden.
@FALSE	**0**	Specified control does not exist.
@NORMAL	**2**	Specified control is visible.

This function returns the current state of the window or control specified by the handle. If a window handle is specified, cWndState is similar to the WIL *WinState* except that it accepts a window handle as the parameter instead of a partial window name.

Example:

```
AddExtender("wwctl34i.dll")
sAppName=  "Notepad.exe"
sAppWindow = "~Notepad"
sAppClass = "NotePad"

; Find Notepad
TestApp = FileLocate (sAppName)
if TestApp == "" then goto Error

; Start notepad
if !ShellExecute(TestApp, "", "", @ICON, "") then goto Error
if !WinWaitExist(sAppWindow, 16)  then goto Error

hWnd  = cFindClass(sAppClass)
if !hWnd then goto Error
```

```
; Did it start minimized?
if cWndState( hWnd ) == @ICON
    Message( Success,  Notepad is minimized.)
    goto done
endif

:Error
message("Error!", "This script didnt work.")

:done
if WinExist(sAppWindow)
   Winclose(sAppWindow)
endif
```

See Also:

DllHwnd, cFindClass

RoboScripter Tutorial

What is the Control Manager Extender?

Overview of Control Manager Usage

Types of Controls

This 'RoboScripter Tutorial' is intended to be a step by step guide to aid in the understanding of how the Control Manager Extender, and the RoboScripter, are used to generate code for a WIL script.

How to enter text into an EditBox control

This tutorial is going to teach you how to write text into the EditBox (control) of Notepads 'File | Open' dialog.

Step 1:
Run the RoboScripter. To launch RoboScripter select 'Start | Programs | WinBatch | RoboScrp w32…

Step 2:
You will then be prompted to run the program you want to automate. Locate Notepad.exe in your windows directory, then press 'Open'. At this point Notepad and RoboScripter should be launched and on the screen. (if not, please repeat steps 1 and 2)

The following is an example RoboScripter dialog. Click on the controls in the dialog for pop-up help.

Step 3:
Now that Notepad is launched and on the screen, we need to bring up the 'File | Open' dialog. This can be done, by dragging the 'Select Window' cross-hair control over the Notepad window's title bar.

Step 4:
You will then see a set of 'actions' appear in the RoboScripter Dialog. Select the SendKey radio button. An edit control will appear, that allows you to type the appropriate keystrokes.

```
Type:  !fo
```

Press the 'Perform' button to perform the operation specified.

At this point you should see a dialog, with the window title 'Open' on the screen. (if not, please repeat all previous steps)

Step 5:
Now that Notepads 'Open' dialog is on the screen, drag and drop the 'Select Window' cross-hairs on the 'File name' editbox control. You will then see a set of

'actions' appear in the RoboScripter Dialog. Select the 'cSetEditText' radio button. An Edit control will appear in the RoboScripter dialog, that allows you to type the file path. Type the name of an existing file (i.e., 'c:\autoexec.bat')

Press the 'Perform' button to perform the operation specified.

Step 6:

Next, drag and drop the 'Select Window' cross-hairs on the 'Open' button control. You will then see a set of 'actions' appear in the RoboScripter Dialog. Select the 'cClickButton" radio button.

Press the 'Perform' button to perform the operation specified.

At this point you should see the file loaded into Notepad. (if not, please repeat each previous step)

Step 7:

We are nearing completion. Select the 'Paste' button on the RoboScripter Dialog. This will paste the buffered code into the Windows Clipboard. Next, select 'Quit', to close RoboScripter.

Step 8:

Open Winbatch Studio.exe, then open a new file.

```
Type:  ctrl-v
```

This will paste the buffered code into your Winbatch script.

Completed script

```
;RoboScripter
; Made with
;   RoboScripter ver: 28
;   CtlMgr ver: 20025
AddExtender("wwctl34i.dll")

;Launch Program
Run("C:\WINDOWS\Notepad.exe","")

;Title: Untitled - Notepad
;ID: 2548
;Class: Notepad
;Level: 1
ControlHandle=cWndByWndSpec("Notepad","NOTEPAD",1,15)
cSetFocus(ControlHandle)  ; Activates Window
TimeDelay(1)
SendKey(`!fo`)     ;Sends Keystrokes


;Title: <untitled>
;ID: 1152
;Class: Edit
;Level: 2
window1=cWndByWndSpec("#32770","NOTEPAD",13,1091,1137,1
088,1120,1121,1090,1152,1089,1136,1040,1,2,1038)
ControlHandle=cWndByID(window1,1152)
cSetEditText(ControlHandle,`c:\autoexec.bat`)


;Title: &Open
;ID: 1
```

```
;Class: Button0
;Level: 2
window1=cWndByWndSpec("#32770","NOTEPAD",13,1091,1137,1
088,1120,1121,1090,1152,1089,1136,1040,1,2,1038)
ControlHandle=cWndByName(window1,`Open~`)
cClickButton(ControlHandle)
```

RoboScripter Tips and Tricks

What is the Control Manager Extender?

Overview of Control Manager Usage

Types of Controls

Except for very simple applications, RoboScripter probably will not be able to write the exact script you need. However it can get close, and do much of the heavy lifting required to successfully use Control Manager based scripts.

In using RoboScripter to write various scripts, a few points have been noticed, and by relating them to you, perhaps we can save you the time required to discover them yourself.

a) Some applications need a DirChange to a desired "working" or "Start In" directory before being run. In this case add

```
DirChange("C:\some\desired\directory")
```

before the RUN statement in your script.

b) RoboScripter is unable to record the click on a top-level menu item and a subsequent click on a drop down menu item. To select a menu item from an application, drag the crosshairs to the application title bar, then select the SENDKEY option to send the menu hot-key characters to the application instead.

c) The full power of the Control Manager extender is not currently supported by RoboScripter. In addition you may encounter strange Window or control types that confuse RoboScripter.

In this case a good workaround is to drag the cross-hairs to the control and select the cSetFocus option, which will activate the control. Later you can edit the script to add special code there, SendKeys perhaps, to deal with the control.

Control Manger Analysis Tutorial

This 'Control Manger Analysis Tutorial' is intended to be a step by step guide to aid in the understanding of how the Control Manager Extender, and its functions, are incorporated into a WIL script.

Tutorial One

Tutorial Two

This tutorial will cover….

What is the Control Manager Extender?

The Control Manager Extender is a WIL extender DLL that allows near-complete access to all standard Windows controls displayed on the screen, and especially within Dialog Boxes presented by various applications.

Control Manger Extender DLL Name: WWCTL34i.DLL

Each function has a complete running example in the Control Manager help file, to show its usage.

The basic technique is to open the application and get the dialog boxes you are interested in displayed on the screen.

You can either:

Run the RoboScripter to generate the code, via a wizard like interface.

Or

Run the Window Analysis script to create a text file detailing the contents of the window in question. Each window and control has its own unique window handle (often seen as hWnd or Wnd). Use the output of the Window Analysis script and the various cWndBy... functions in this extender to obtain the handle to the window or control you need, then use the specific functions designed for each type of control to either obtain information about the item, or to set it to the desired value.

What is the Control Manager Extender?

Overview of Control Manager Usage

The most important part is to understand how to either:

Use the RoboScripter,which is included in the Control Manager ZIP file, to dynamically generate the code

Or

Run the "Window Analysis" script (ANALYSIS.WBT), which is included in the Control Manager ZIP file.

First get the window that you want to control up on the screen. Next run the ANALYSIS.WBT script in another window, which when you select the title of the window you want to identify, will produce a report showing the unique control identifier IDs, names, and sequence numbers. You can then use this information in the functions included with the extender.

Background Information:
Basically all windows exist in a tree-structured list off a "parent" window. Each and every window (which includes buttons, radio buttons, check boxes, lists and *everything*) has an absolutely unique window handle. However the window handles change every time you run the program.

To do any thing with a window you first need the window handle. But as it is totally unpredictable, you need a way to determine the window handle when you need it.

Beside the handle, which is unique, windows have a number of other attributes which can be used to identify it. However these other attributes are not always unique, so you must pick and choose among them to decide what to do.

These other attributes are the windows name or title, the position in the list of windows (known as the sequence number), the type of window (known as the class) and the window identifier (known as the ident or ID).

By viewing the report produced by the "Window Analysis" script, you can determine some way to work down from the parent top windows (which is the window handle returned by DllHwnd) to the exact window you wish to modify.

You need to work down from the parent to the desired lower level "child" window. Use the cWndByxxx functions to do this.

Listed by ease-of-use (easiest/best choice to most complex):
cWndById
cWndByName
cWndByClass
cWndBySeq

First, I see if a window at a particular level has a unique ID. If so I prefer to use that. If two windows at the same level have the same ID, then I see if the window names (or titles) are different. If not, then I check the class of the windows. And

finally resort to the sequence number of the window.

I don't like sequence numbers as they can quickly and mysteriously change when you least expect it.

We've added a whole slew of new functions recently, which can get at the more complex types of standard windows controls.

Types of controls …
button
radiobutton
checkbox
combobox
listbox

Types Of Controls

Controls that can be manipulated by the Control manager functions include (Click on name of control to see a graphic example of that control):

Check Boxes- check boxes can be interrogated, checked, or unchecked.

Radio Buttons - radio buttons can be interrogated or set.

Tab/Tabbed Dialogs - a particular tab in a Tabbed Dialog can be selected.

List Boxes and **Combo Boxes** - contents of List and Combo Boxes can be interrogated and specific items within the List or Combo Box can be selected.

Edit Boxes - set or retrieve contents of an Edit Box.

Status Bar - (msctls_statusbar32) retrieve text that may be associated with each part of a Status Bar control.

Track Bar - (msctls_trackbar32) retrieve or set position of a Track Bar control.

UpDown - (msctls_updown32) retrieve or set position of a UpDown control.

Button/PushButton - click a button.

Date Time Picker - (SysDateTimePick32) retrieve the date and time stored in a Date Time Picker control in standard datetime format.

Header - (SysHeader32) retrieve all the text associated with each item of a Header common control.

IP Address - (SysIPAddress32) retrieve or place an IP address stored in an IP Address common control.

List View - (SysListView32) retrieve text, un-select or select an item of a List View common control.

Month Calendar - (SysMonthCal32) retrieve or interrogate the date or date range of a Month Calendar control.

Tree View - (SysTreeView32) un-select or select an item of a Tree View common control.

ToolBar - (ToolbarWindow32) select or retrieve text for each button of a Tool Bar control.

Tutorial One

How to enter text into an EditBox control

This tutorial is going to teach you how to write text into the EditBox (control) of Notepads 'File | Save As' dialog.

Step 1:

Before you begin writing the script, you will need to bring the dialog you are interested in interrogating, up on the screen. To launch Notepad, simply go to the Start Menu, select Run, and type notepad.exe. At this point Notepad should be launched and on the screen. (if not, please repeat step one)

Step 2:

Now that Notepad is launched and on the screen, we need to bring up the 'File | Save As' dialog. This can be done, by selecting the File menu, then select 'Save As' from the menu. At this point you should see a dialog, with the window title 'Save As' on the screen. (if not, please repeat steps one and two)

Step 3:

Now that Notepads 'Save As' dialog is on the screen, we are ready to interrogate the dialog with the Windows Analyis Script.

What does the Windows Analysis script do?

The Windows Analysis script (analysis.wbt) creates a report (trash.txt) on the windows visible at the time the report is run.

The basic technique is to open the application and get the dialog boxes you are interested in displayed on the screen.

The Window Analysis script will create / output a text file detailing the contents of the window in question.

How do I use it?

Run the script, by double clicking on analysis.wbt (which is most likely located in the WinBatch\Samples subdirectory).

A dialog with all the visible window titles, will appear. Choose the window title 'Save As' from the dialog, then press Ok. The Window Analysis script will process and output, a file called 'trash.txt' and load it into Notepad for you to view.

It will look some thing like this…….

```
P   C   C   C   C
A   H   H   H   H
R   I   I   I   I
E   L   L   L   L
N   D   D   D   D
T   2   3   4   5    CLASS                      IDENT  TITLE
-------------------------------------------------------------------------------
TOP                  #32770                     0      Save As
```

```
1                   Static                  1091    Save &in:
    X               NONE

2                   ComboBox                1137
    X               NONE

3                   Static                  1088
    X               NONE

4                   ListBox                 1120
    X               NONE

5                   SHELLDLL_DefView        1121
    1               SysListView32           1
        1           SysHeader32             0
            X       NONE

6                   Static                  1090    File &name:
    X               NONE

7                   Edit                    1152
    X               NONE

8                   Static                  1089    Save as &type:
    X               NONE

9                   ComboBox                1136
    X               NONE

10                  Button                  1040    Open as &read-only
    X               NONE

11                  Button                  1       &Save
    X               NONE

12                  Button                  2       Cancel
    X               NONE

13                  Button                  1038    &Help
    X               NONE

14                  ToolbarWindow32         1
    X               NONE
```

Step 4:

At this point the Windows Analysis' output, trash.txt, should be on the screen. (if not, please repeat step three)

Explanation of the Windows Analysis script output…

Information is listed by columns.

The first five columns (PARENT, CHILD2, CHILD3, CHILD4, CHILD5) will tell you where a particular control falls in the hierarchical scheme of things. Meaning, relationship of parent-to-child, and their sequential order.

The next column is the CLASS column. This column lists the 'types' of controls.

The IDENT column lists each individual controls identifer. In other words, an id number that is associated to that particular control.

The TITLE column lists the title of the control, if any.

Step 5:

Now, that we understand the structure of the Windows Analysis script output, a little better….

For this example, we are interested in two things, the top level parent and EditBox Control.

You will always need to get a handle to a top level parent window first. This can be done using the DLLHwnd function. As follows:

```
parentHwnd=DLLHwnd("Save As")
```

Edit (under the column heading CLASS) is important because that is the control that we want to fill in with the filename into Notepad.

The two important lines in the output should read as follows:

```
P   C   C   C   C
A   H   H   H   H
R   I   I   I   I
E   L   L   L   L
N   D   D   D   D
T   2   3   4   5    CLASS                    IDENT  TITLE
-------------------------------------------------------------------------------
TOP                  #32770                   0      Save As
    7                Edit                     1152
```

TOP (Top level parent)
The very first row of any Window Analysis script output should always have the value TOP under the PARENT column. As you see above.

EDIT
You will notice the number 7, in this case, falls under the column 'CHILD2'. What this tells you, is that the edit box control is the 7th sequential control and it is a second level child.

The CLASS is Edit, which is the same as the type EditBox.

The IDENT number is 1152. And there is no TITLE, in this case.

It is important to know, that this control falls under the CHILD2 column because, it tells us how far we need to 'drill down' to get the handle to the control. This will be explained more thoroughly at a later point.

Step 6:

Lets begin to write the script.

We are using the Control Manager extender, therefore we need to include the following statement, that adds the extender:

```
AddExtender("WWCTL34i.DLL")
```

As mentioned previously, we first need to get a handle to the top level

parent window. This can be done as follows:

```
parentHwnd=DLLHwnd("Save As")
```

Notice: the window TITLE, we handed to the function DLLHwnd is EXACTLY the same as, it is output by the Windows analysis script.

We now need to 'drill-down' to the control, using the cWndby__ functions.

What is meant by drill-down?
It means that we need to get a handle to every 'top-level' control, starting with the top level parent and working our way down through the child hierarchy. You can work down from the parent to the desired lower level "child" window by using the cWndBy__ functions to do this.

Listed by ease-of-use (easiest/best choice to most complex):
cWndById
cWndByName
cWndByClass
cWndBySeq

First, I see if a window at a particular level has a unique ID. If so I prefer to use that. If two windows at the same level have the same ID, then I see if the window names (or titles) are different. If not, then I check the class of the windows. And finally resort to the sequence number of the window.

Since the control we are attempting to get the handle to, is at the level CHILD2 and has a distinctive IDENT, we can then simply use the cWndbyID function to retrieve the handle to the edit box control. This can be done as follows:

```
editHwnd=cWndByID(parentHwnd, 1152)
```

We now have a handle to the editbox control. Using the function cSetEditText we can fill in the control with the file name, we would like to save the file as…

```
cSetEditText(editHwnd, "C:\Windows\Desktop\Test.txt")
```

At this point your code should read as follows:

```
AddExtender("wwctl34i.dll")
parentHwnd=DllHwnd("Save As")
editHwnd=cWndByID(parentHwnd,1152)
cSetEditText(editHwnd, "C:\Windows\Desktop\Test.txt")
```

Step 7:

Lets add to the script.

At this point we have the basic code that will get a handle to the 'Save As' dialog, then it gets a handle to the edit control and finally put the line 'C:\Windows\Desktop\Test.txt' into the edit control.

There are a few more things we may want to add. For instance, we probably want our WinBatch script to launch Notepad, wait for the window to appear and bring up the 'Save As' dialog, automatically. The code to do this, looks like the following:

```
Run("notepad.exe","")
SendMenusTo("~Notepad","FileSaveAs")
WinWaitExist("Save As",5)
```

We have now completed the script. Here is what the completed script should look like:

Completed script

```
AddExtender("wwctl34i.dll")
Run("notepad.exe","")
SendMenusTo("~Notepad","FileSaveAs")
WinWaitExist("Save As",5)
parentHwnd=DllHwnd("Save As")
editHwnd=cWndByID(parentHwnd,1152)
cSetEditText(editHwnd, "C:\Windows\Desktop\Test.txt")
Message("Control Manager Example","Done")
exit
```

Tutorial Two

How to select an 'item' in the Windows Explorer.

This tutorial is going to teach you how to select an item in a SysListView32 (control) in the Windows Explorer.

Step 1:

Before you begin writing the script, you will need to bring the dialog you are interested in interrogating, up on the screen. To launch Explorer, simply go to the Start Menu, select Run, and type explorer.exe. At this point Explorer should be launched and on the screen. (if not, please repeat step one)

Step 2:

Now that Explorer is launched and on the screen, we are ready to interrogate the Explorer window (dialog) with the Windows Analysis Script.

What does the Windows Analysis script do?

The Windows Analysis script (analysis.wbt) creates a report (trash.txt) on the windows visible at the time the report is run.

The basic technique is to open the application and get the dialog boxes you are interested in displayed on the screen.

The Window Analysis script will create / output a text file detailing the contents of the window in question.

How do I use it?

Run the script, by double clicking on analysis.wbt (which is most likely located in the WinBatch\Samples subdirectory).

A dialog with all the visible window titles, will appear. Choose the window title 'Exploring - C:\' from the dialog, then press Ok. The Window Analysis script will process and output, a file called 'trash.txt' and load it into Notepad for you to view.

It will look some thing like this…….

```
P   C   C   C   C
A   H   H   H   H
R   I   I   I   I
E   L   L   L   L
N   D   D   D   D
T   2   3   4   5    CLASS                    IDENT  TITLE
-------------------------------------------------------------------------------
TOP                  ExploreWClass            3764   Exploring - C:\

    1                WorkerA                  40965
        1            ReBarWindow32            40965
            1        ToolbarWindow32          0
                X    NONE
            2        ComboBoxEx32             41477
```

```
                1    ComboBox                  41477
                1    Edit                      41477
                X    NONE
           3         ToolbarWindow32           40960
                X    NONE
           4         WorkerA                   0
                X    NONE
           5         ToolbarWindow32           0
                X    NONE

2                    WorkerA                   9999
    X                NONE

3                    msctls_statusbar32        40961
    X                NONE

4                    SHELLDLL_DefView          0
    1                SysListView32             1
           1         SysHeader32               0
                X    NONE

5                    BaseBar                   0
    1                ReBarWindow32             40965
           1         ToolbarWindow32           0
                X    NONE
           2         SysTreeView32             40963
                X    NONE
```

Step 3:

At this point the Windows Analysis' output, trash.txt, should be on the screen. (if not, please repeat step two)

Explanation of the Windows Analysis script output…

Information is listed by columns.

The first five columns (PARENT, CHILD2, CHILD3, CHILD4, CHILD5) will tell you where a particular control falls in the hierarchical scheme of things. Meaning, relationship of parent-to-child, and their sequential order.

The next column is the CLASS column. This column lists the 'types' of controls.

The IDENT column lists each individual controls identifier. In other words, an id number that is associated to that particular control.

The TITLE column lists the title of the control, if any.

Step 4:

Now, that we understand the structure of the Windows Analysis script output, a little better….

For this example, we are interested in three main things, the top level parent, SHELLDLL_DefView and SysListView32 Control.

You will always need to get a handle to a top level parent window first. This can be done using the DLLHwnd function. As follows:

```
parentHwnd=DLLHwnd("Exploring - C:\")
```

SHELLDLL_DefView (under the column heading CLASS) is important

because we will need to get a handle to it inorder to drill down to the SysListView32 control.

SysListView32 (under the column heading CLASS) is important because that is the control that we want to get the information from.

The three important lines in the output should read as follows:

```
P   C   C   C   C
A   H   H   H   H
R   I   I   I   I
E   L   L   L   L
N   D   D   D   D
T   2   3   4   5    CLASS                    IDENT  TITLE
-------------------------------------------------------------------------------
TOP                  ExploreWClass            3764   Exploring - C:\
    4                SHELLDLL_DefView         0
        1            SysListView32            1
```

TOP (Top level parent)
The very first row of any Window Analysis script output should always have the value TOP under the PARENT column. As you see above.

The following items will depend on what window is being analyzed.

The CLASS, in this case, is ExploreWClass.

The IDENT number , in this case, is 3764.

The TITLE , in this case, is 'Exploring - C:\'. The TITLE is what is passed to the **DllHwnd** function

SHELLDLL_DefView
You will notice the number 4, in this case, falls under the column 'CHILD2'. What this tells you, is that the SHELLDLL_DefView class is the 4th sequential control and it is a second level child.

The CLASS is SHELLDLL_DefView.

The IDENT number is 0. And there is no TITLE, in this case.

SysListView32
You will notice the number 1, in this case, falls under the column 'CHILD3'. What this tells you, is that the SysListView32 control is the 1st sequential control and it is a third level child.

The CLASS is SysListView32.

The IDENT number is 1. And there is no TITLE, in this case.

It is important to know, that this control falls under the CHILD3 column because, it tells us how far we need to 'drill down' to get the handle to the control.

We will need to get a handle to the Top level window first, using DLLHwnd. We will then need to get a handle to the CHILD2 control, and finally we will get the handle to the CHILD3 control, which is the SysListView32 control we want to interrogate.

Step 5:

Lets begin to write the script.

We are using the Control Manager extender, therefore we need to include the following statement, that adds the extender:

```
AddExtender("WWCTL34i.DLL")
```

As mentioned previously, we first need to get a handle to the top level parent window. This can be done as follows:

```
TOPhwnd=DLLHwnd("Exploring - C:\")
```

Notice: the window TITLE, we handed to the function DLLHwnd is EXACTLY the same as, it is output by the Windows analysis script.

We now need to 'drill-down' to the control, using the cWndby__ functions.

What is meant by drill-down?
It means that we need to get a handle to every 'top-level' control, starting with the top level parent and working our way down through the child hierarchy. You can work down from the parent to the desired lower level "child" window by using the cWndBy__ functions to do this.

Listed by ease-of-use (easiest/best choice to most complex):
cWndById
cWndByName
cWndByClass
cWndBySeq

First, I see if a window at a particular level has a unique ID. If so I prefer to use that. If two windows at the same level have the same ID, then I see if the window names (or titles) are different. If not, then I check the class of the windows. And finally resort to the sequence number of the window.

We will now need to get a handle to the CHILD2 control, before we attempt to get a handle to the CHILD3 control. Since the CHILD2 control is of the distinctive class SHELLDLL_DefView, we can use the function cWndByClass…

```
CHILD2hwnd = cwndbyclass(TOPhwnd,"SHELLDLL_DefView")
```

Since the control we are attempting to get the handle to, is at the level CHILD3 and has a distinctive IDENT, we can then simply use the cWndbyID function to retrieve the handle to the SysListView32 control. This can be done as follows:

```
;Gets a handle to the SysListView32 control,
;which IDENT happens to be 1.
CHILD3hwnd   = cwndbyid(CHILD2hwnd,1)
```

We now have a handle to the SysListView32 control. Using the function cGetLVText we can get a tab-delimited list of all the items in the SysListView32 control.

```
; Get the labels from the first column of
; each row in the control.
wintext =  cGetLVText(CHILD3hwnd)
```

At this point your code should read as follows:

```
AddExtender("wwctl34i.dll")
TOPhwnd = DLLhwnd("Exploring")
CHILD2hwnd   = cwndbyclass(TOPhwnd,"SHELLDLL_DefView")
;Gets a handle to the SysListView32 control, which
IDENT happens to be 1.
CHILD3hwnd   = cwndbyid(CHILD2hwnd,1)

; Get the labels from the first column of each row in
the control.
wintext =  cGetLVText(CHILD3hwnd)
```

Step 6:

Lets add to the script.

At this point we have the basic code that will get a handle to and retrieve the text of a SysListView control, in the Explorer window.

Now say that we want to be able to 'double click on the 'Windows' directory, so that we can view the contents of that directory.

This can be done be locating the string 'Windows' in the list returned by cGetLVText. Here's the code:

```
myitem = "Windows"
;Get the index of the item
select1=ItemLocate(myitem, wintext, @tab)
```

Now, if ItemLocate returns the value 0, we know that the string 'Windows' has NOT been found, and we should display a message that the 'item' was not found.

On the other hand, if ItemLocate returns the value 1, we know that the string 'Windows' has been found, and should now attempt to select it and double click on it.

This could be coded as follows:

```
if select1
   ; Select the item from the list control.
   if csetlvitem(CHILD3hwnd, select1)
     cdblclickitem(CHILD3hwnd, "")
   else
     ; Can't select the item
     message("Item not selected", "Item number
%select1% no longer exists.")
   endif
else
   message("Item not found",  "Item number %select1%
may no longer exist.")
endif
exit
```

Finally, there are a few more things we may want to add. For instance, we probably want our WinBatch script to launch the Explorer, wait for the window to appear, automatically. The code to do this, looks like the following, and should be added to the beginning of the script.:

```
Run("Explorer.exe","")
WinWaitExist("Exploring",10)
```

Completed script

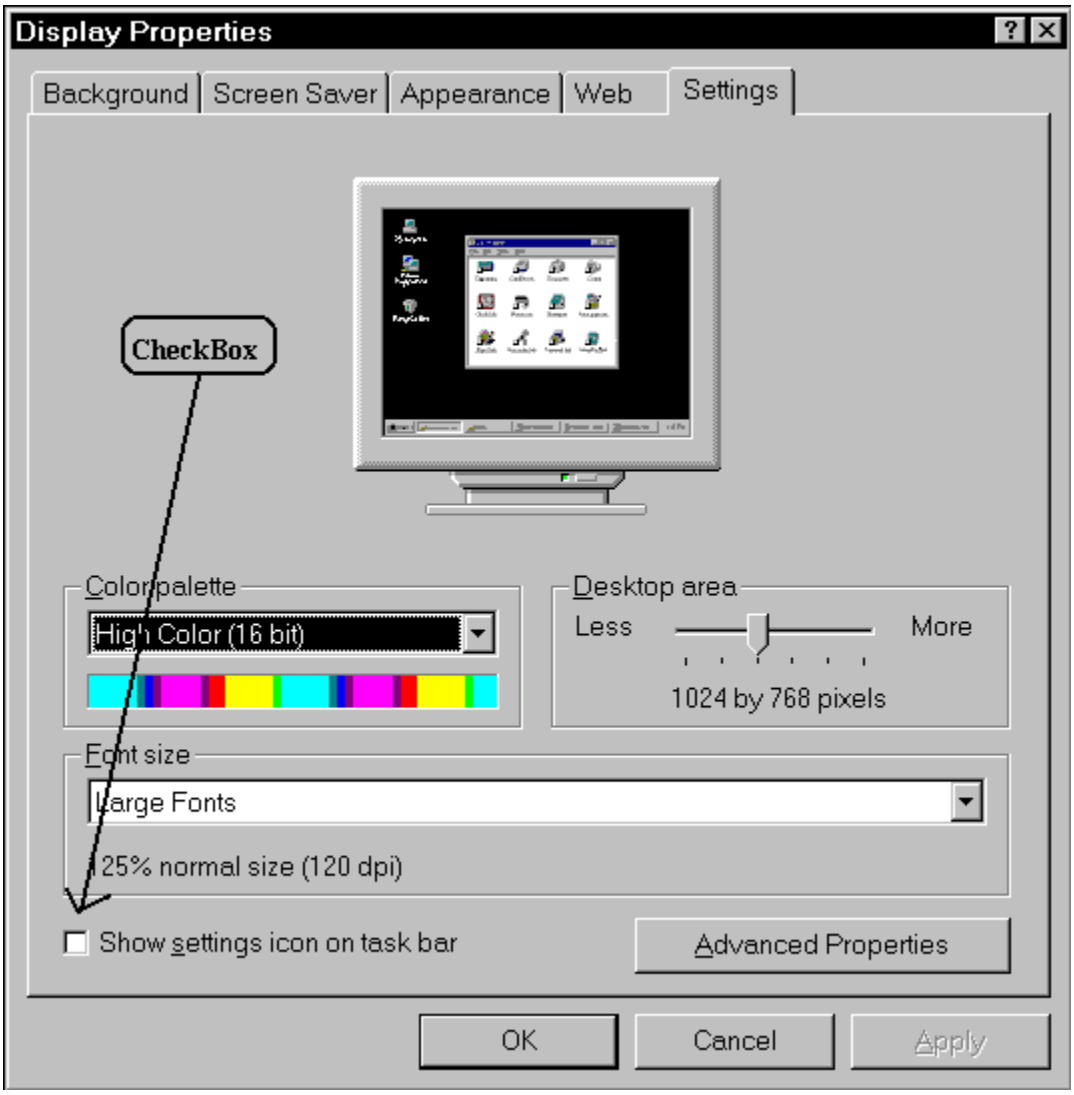
```
AddExtender("wwctl34i.dll")
Run("Explorer.exe","")
WinWaitExist("Exploring",10)
; Explorer List View area
TOPhwnd = DLLhwnd("Exploring")
CHILD2hwnd   = cwndbyclass(TOPhwnd,"SHELLDLL_DefView")
;Gets a handle to the SysListView32 control, which
IDENT happens to be 1.
CHILD3hwnd   = cwndbyid(CHILD2hwnd,1)

; Get the labels from the first column of each row in
the control.
wintext =  cGetLVText(CHILD3hwnd)

myitem = "Windows"
;Get the index of the item
select1=ItemLocate(myitem, wintext, @tab)

if select1
   ; Select the item from the list control.
   if csetlvitem(CHILD3hwnd, select1)
     cdblclickitem(CHILD3hwnd, "")
   else
     ; Can't select the item
     message("Item not selected", "Item number
%select1% no longer exists.")
   endif
else
   message("Item not found",  "Item number %select1%
may no longer exist.")
endif
exit
```

noFALSEyesCTLPOPUPyesyesyesControl Manger PopUp fileyesyes16/02/01

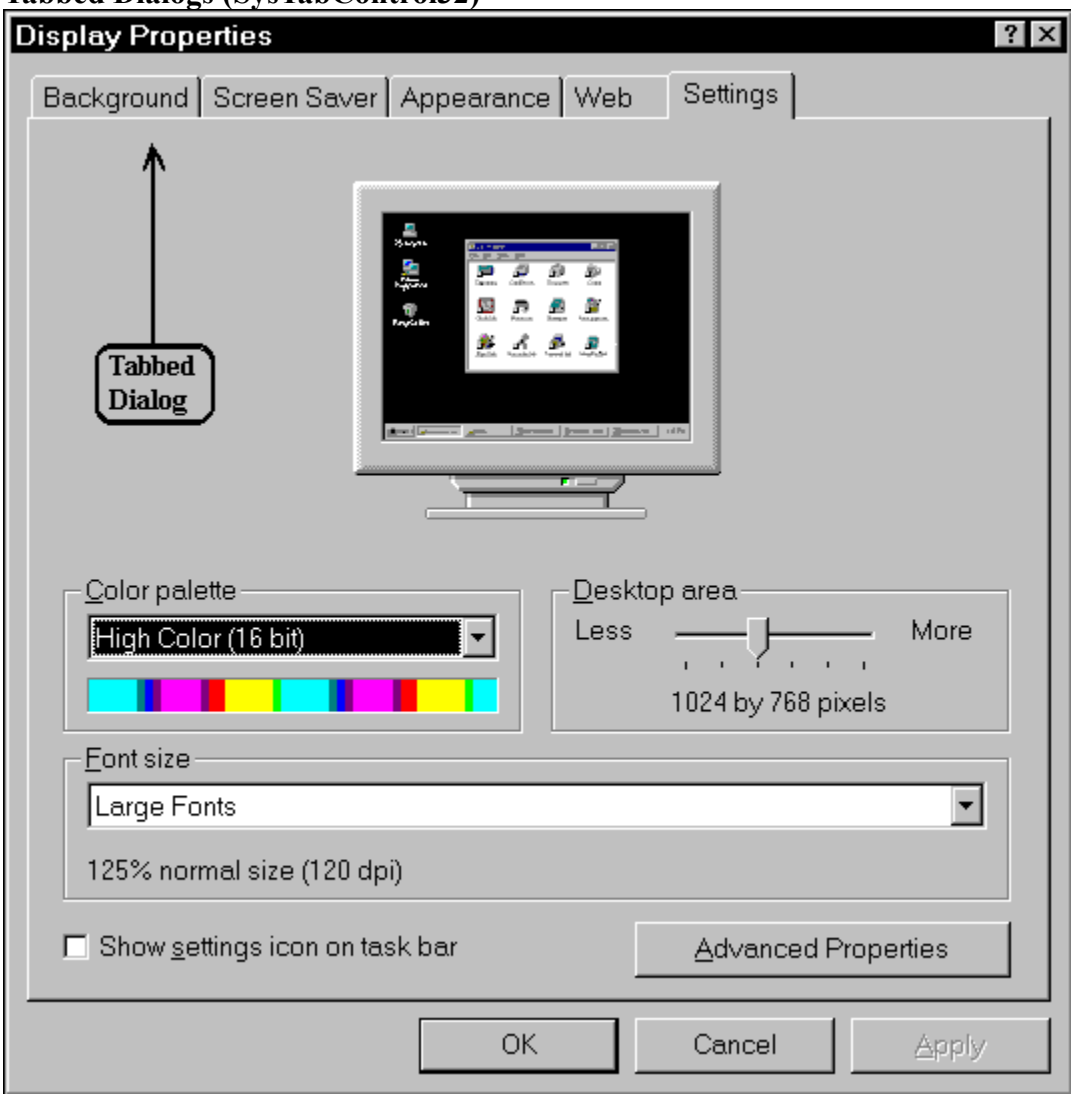
Table of Contents

Help file produced by **HELLLP!** v2.7 , a product of Guy Software, on 02/16/01 for WILSON WINDOWWARE, INC..
The above table of contents will be automatically completed and will also provide an excellent cross-reference for context strings and topic titles. You may leave it as your main table of contents for your help file, or you may create your own and cause it to be displayed instead by using the I button on the toolbar. This page will not be displayed as a topic. It is given a context string of _:_ , but this is not presented for jump selection.
HINT: If you do not wish some of your topics to appear in the table of contents as displayed to your users (you may want them ONLY as PopUps), move the lines with their titles and contexts to below this point. If you do this remember to move the whole line, not part. As an alternative, you may wish to set up your own table of contents, see Help under The Structure of a Help File.
Do not delete any codes in the area above the Table of Contents title, they are used internally by HELLLP!

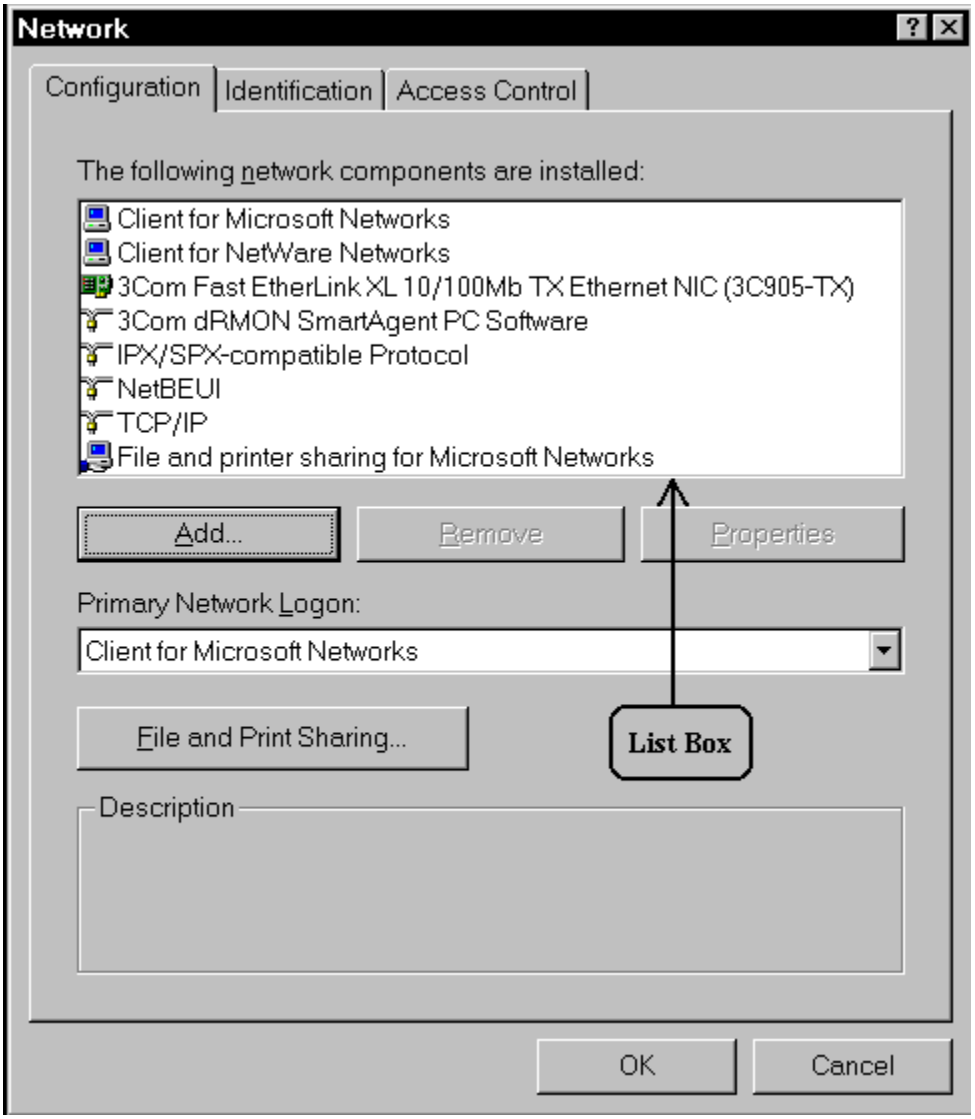
CheckBox


Display Properties
Background
Screen Saver
Appearance
Web
Settings
CheckBox
Color palette
High Color (16 bit)
Desktop area
Less
More
1024 by 768 pixels
Font size
Large Fonts
125% normal size (120 dpi)
Show settings icon on task bar
Advanced Properties
OK
Cancel
Apply

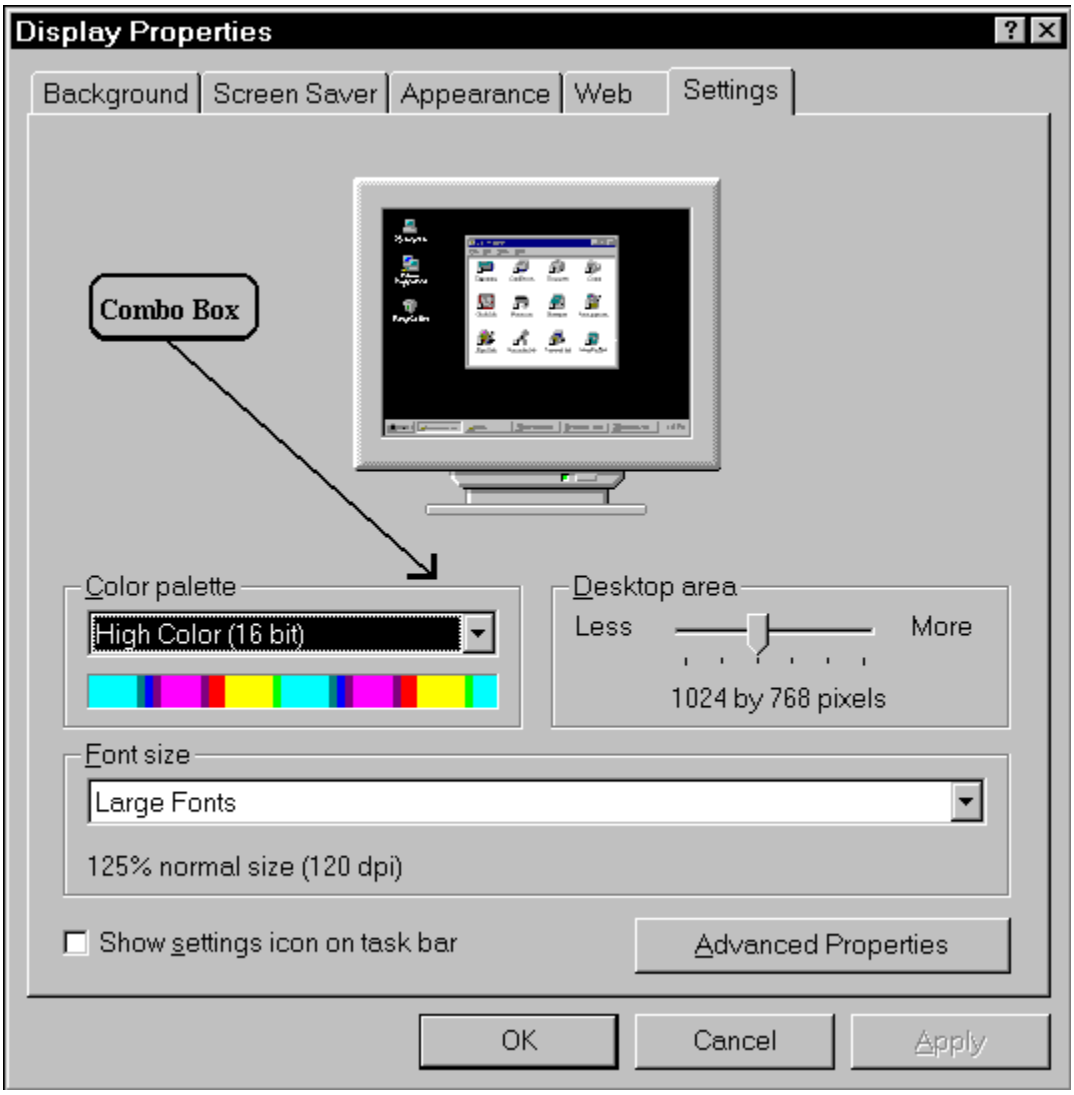
Tabbed Dialogs (SysTabControl32)



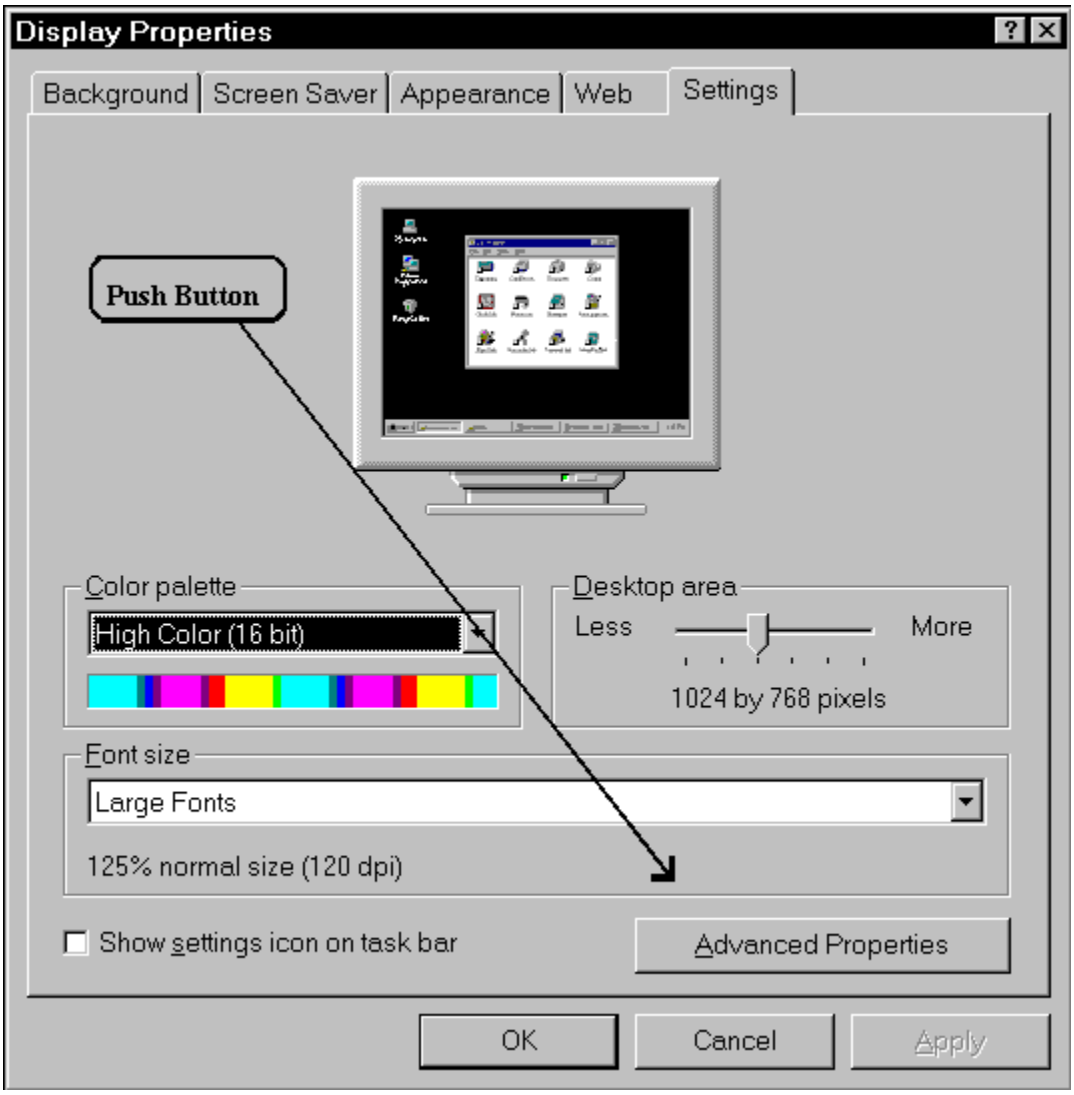
ListBox


Network
Configuration
Identification
Access Control
The following network components are installed:
Client for Microsoft Networks
Client for NetWare Networks
3Com Fast EtherLink XL 10/100Mb TX Ethernet NIC (3C905-TX)
3Com dRMON SmartAgent PC Software
IPX/SPX-compatible Protocol
NetBEUI
TCP/IP
File and printer sharing for Microsoft Networks
Add...
Remove
Properties
Primary Network Logon:
Client for Microsoft Networks
File and Print Sharing...
List Box
Description
OK
Cancel

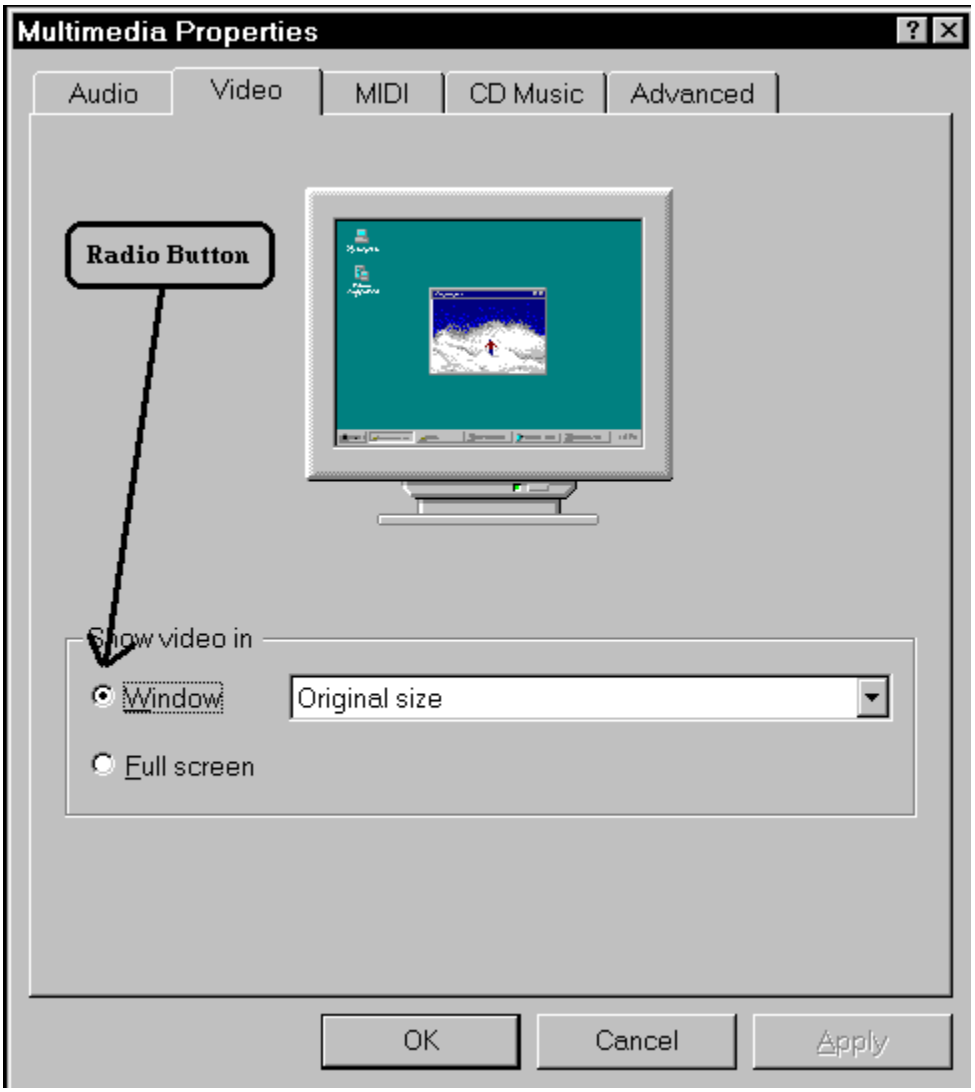
ComboBox


Display Properties
Background
Screen Saver
Appearance
Web
Settings
Combo Box
Color palette
High Color (16 bit)
Desktop area
Less
More
1024 by 768 pixels
Font size
Large Fonts
125% normal size (120 dpi)
Show settings icon on task bar
Advanced Properties
OK
Cancel
Apply

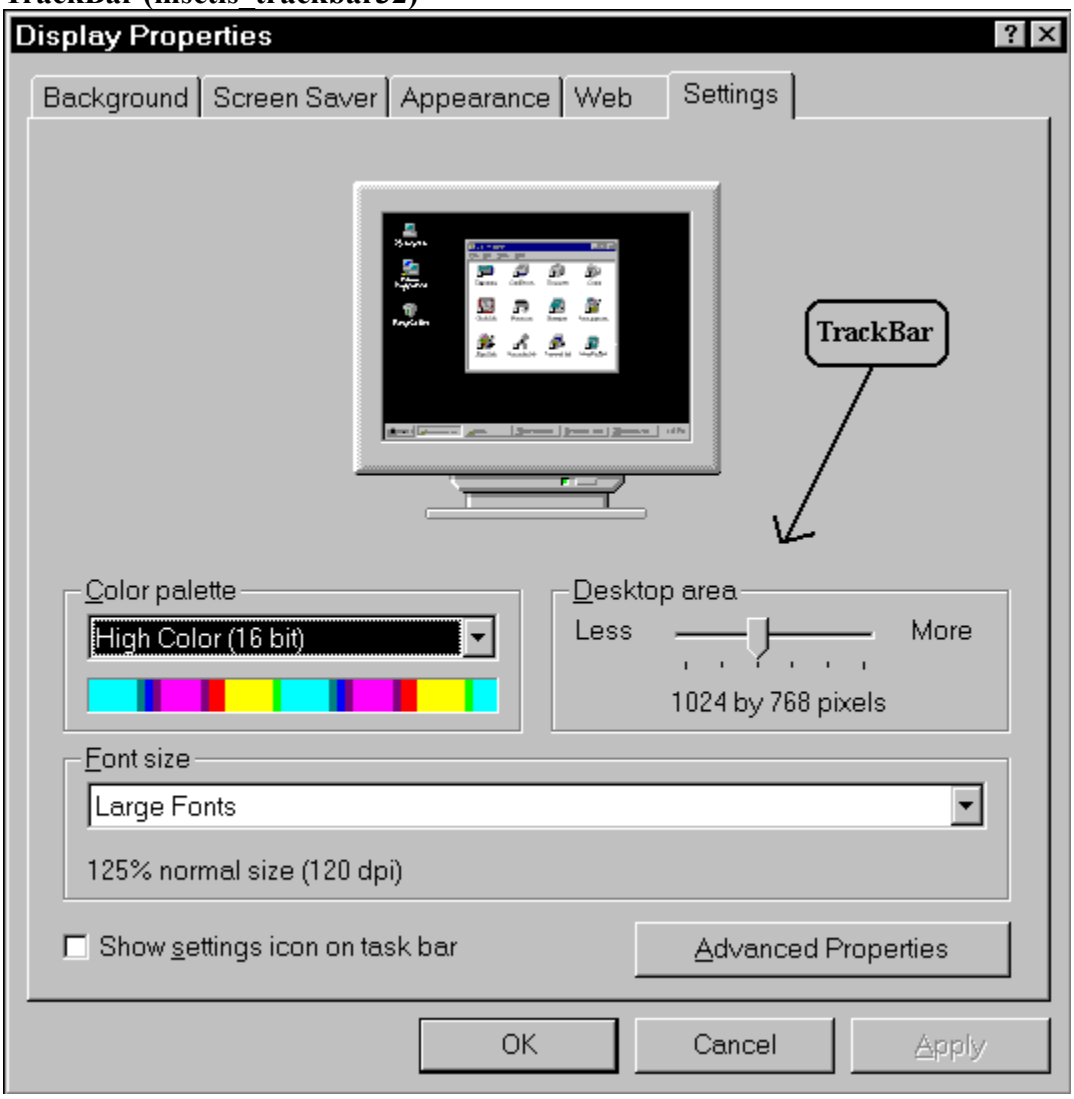
PushButton \ Button



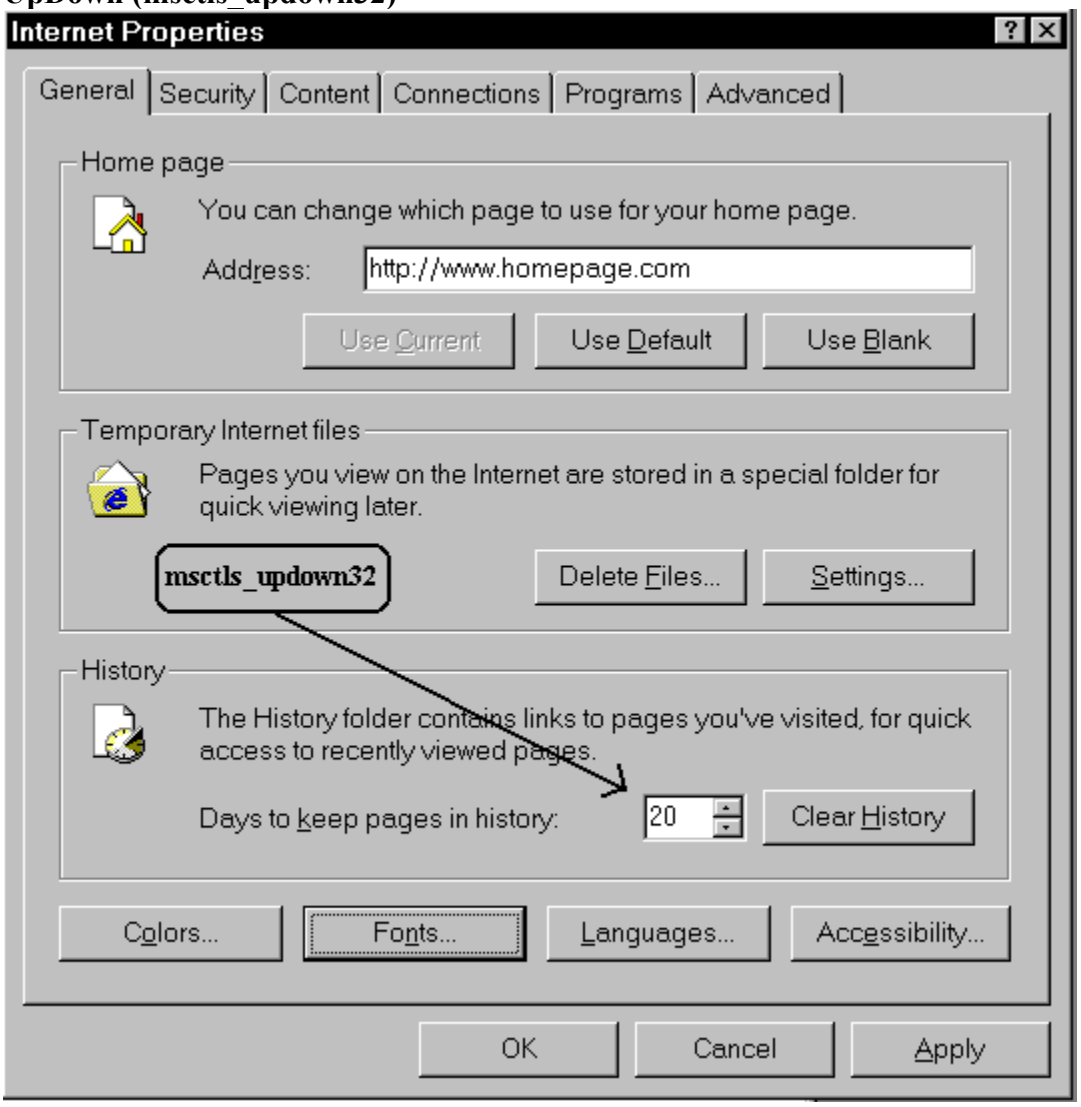
RadioButton



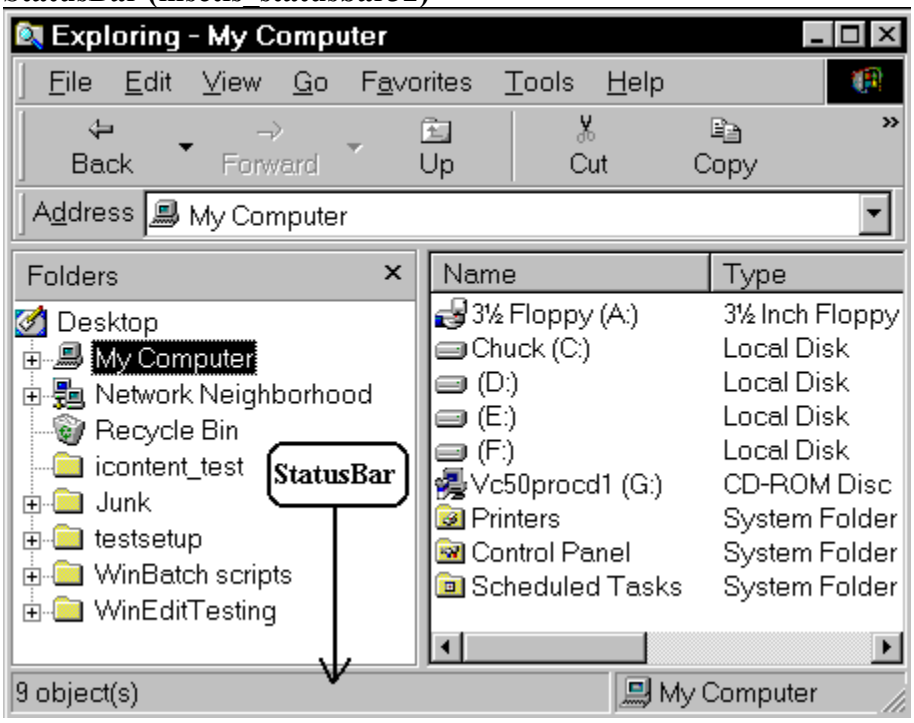
TrackBar (msctls_trackbar32)



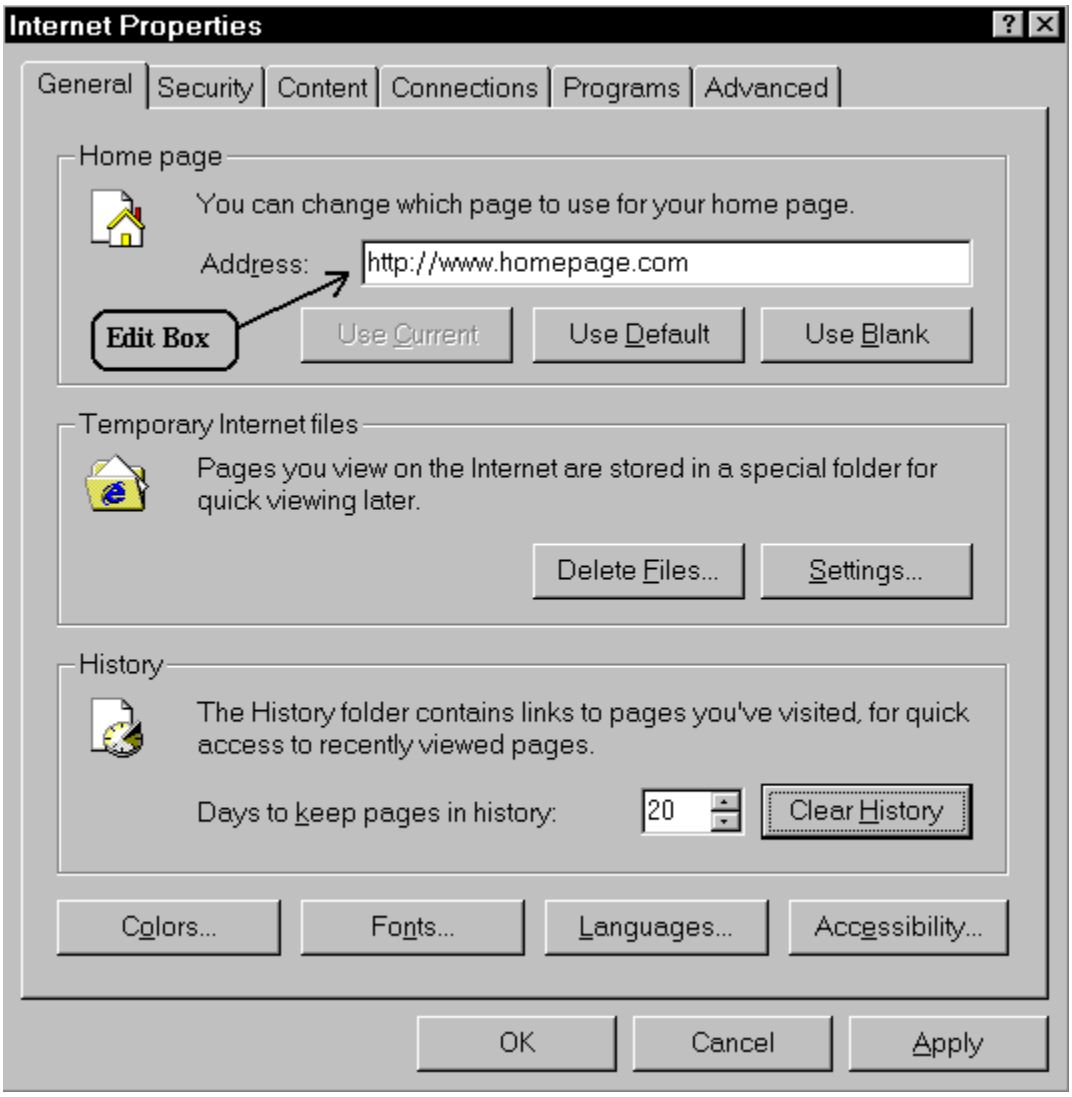
UpDown (msctls_updown32)



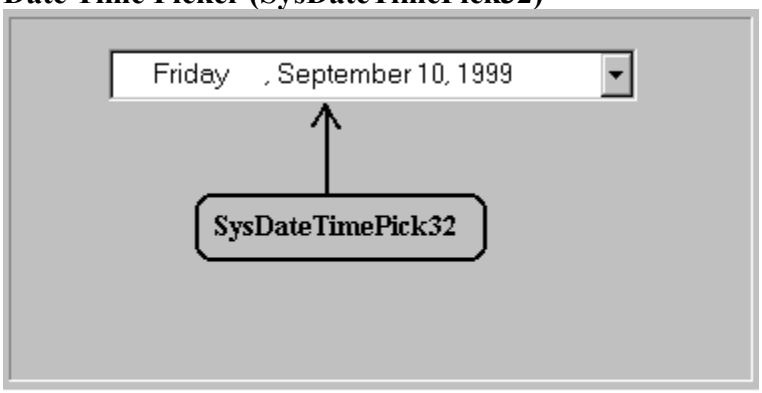
StatusBar (msctls_statusbar32)



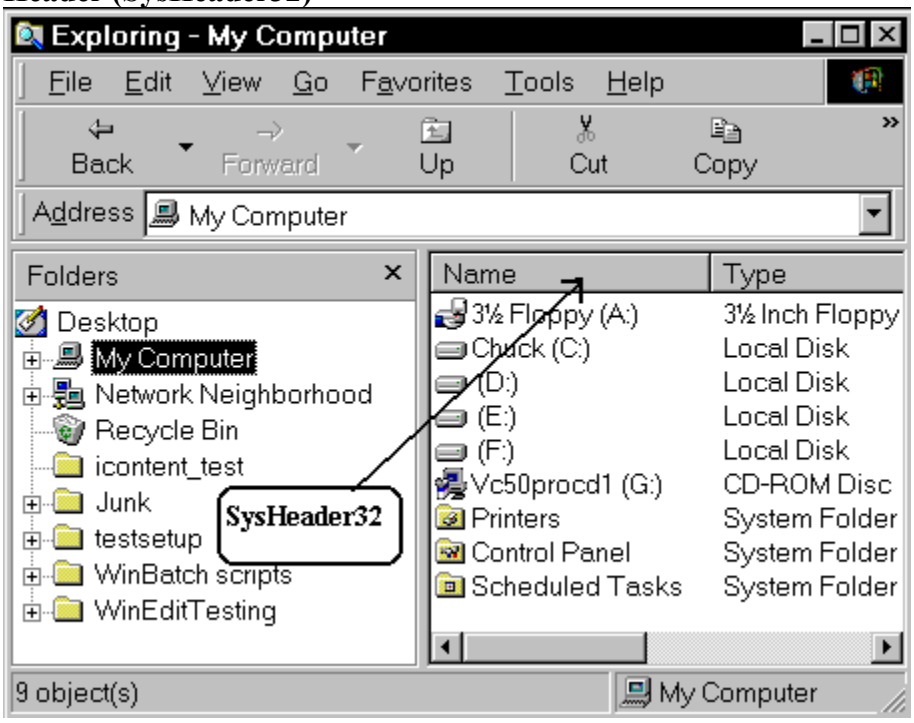
EditBox


Internet Properties
General
Security
Content
Connections
Programs
Advanced
Home page
You can change which page to use for your home page.
Address:
http://www.homepage.com
Edit Box
Use Current
Use Default
Use Blank
Temporary Internet files
Pages you view on the Internet are stored in a special folder for quick viewing later.
Delete Files...
Settings...
History
The History folder contains links to pages you've visited, for quick access to recently viewed pages.
Days to keep pages in history:
20
Clear History
Colors...
Fonts...
Languages...
Accessibility...
OK
Cancel
Apply

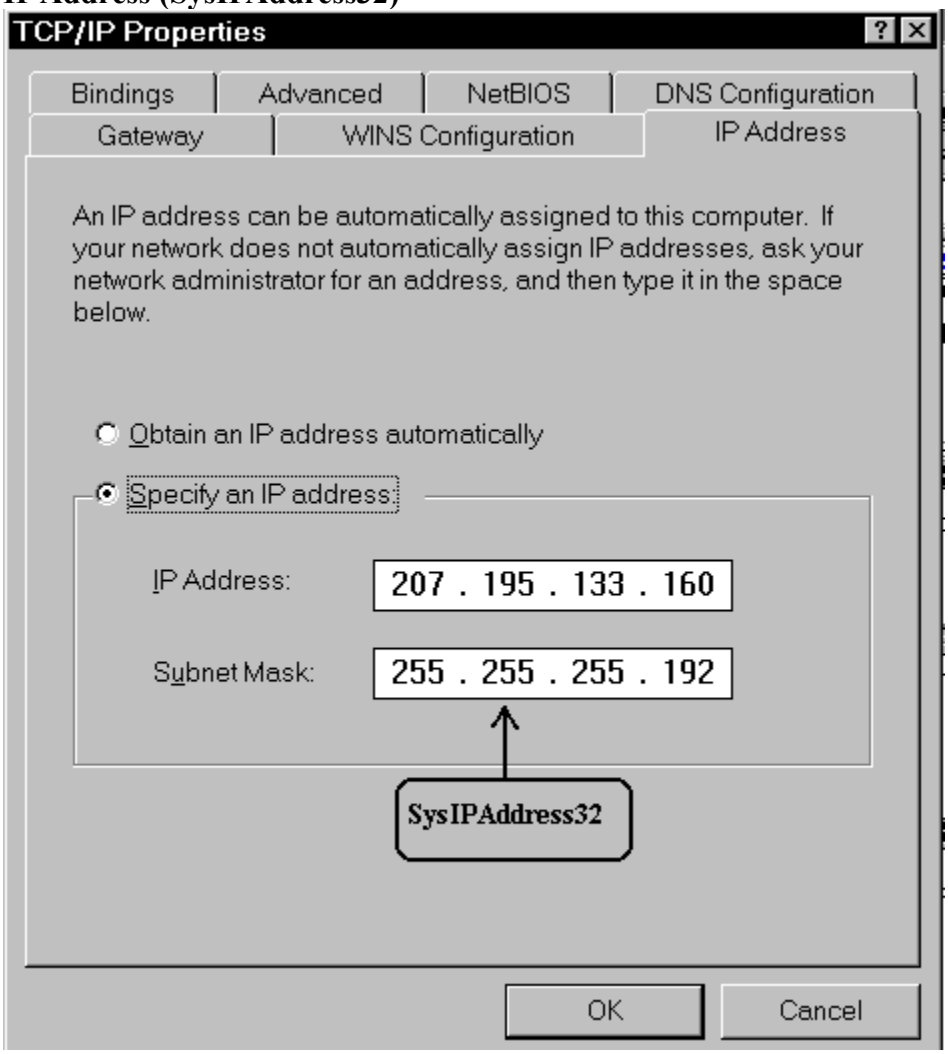
Date Time Picker (SysDateTimePick32)



Header (SysHeader32)



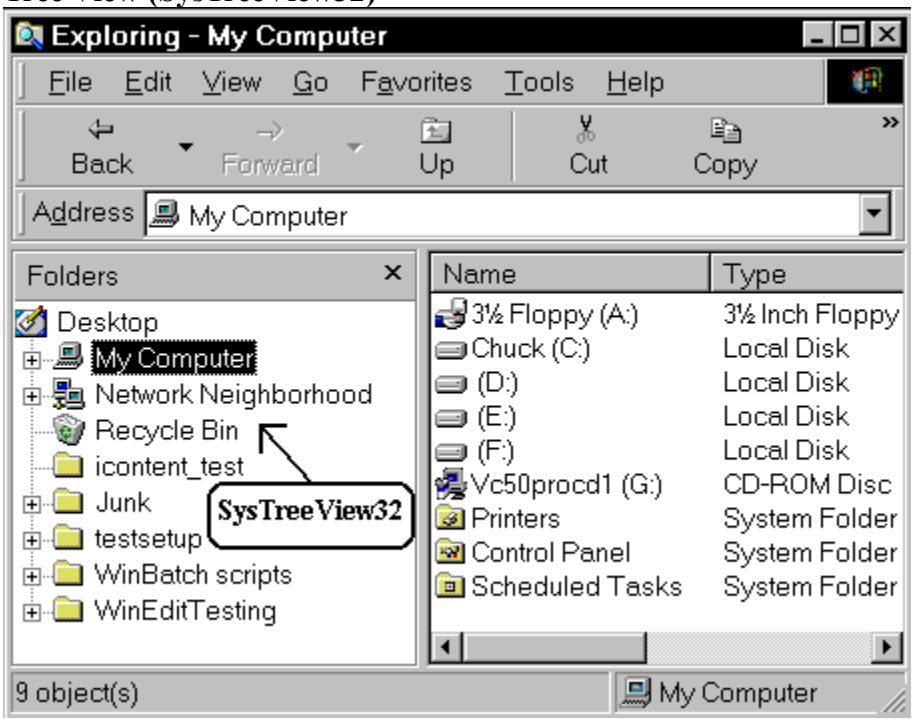
IP Address (SysIPAddress32)



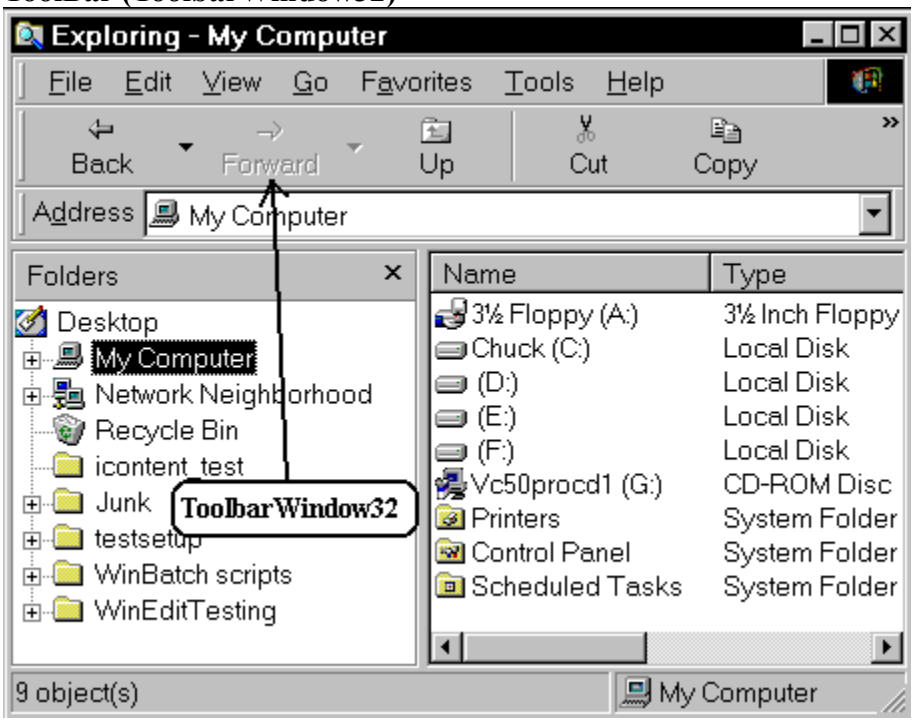
List View (SysListView32)

Month Calendar (SysMonthCal32)

Tree View (SysTreeView32)



ToolBar (ToolbarWindow32)



Window Select Cross-hairs - RoboScripter

Simply drag and drop the cross-hairs into the Window or Control you want to manipulate.

Class Type - RoboScripter

The class type of the window or control.
Example class types: CHECKBOX, COMBOBOX, BUTTON, Etc.

Action - RoboScripter

Actions that can be performed on the selected control. Basically a list of functions that can deal with the selected 'class type' of control.

Perform - RoboScripter

This button will write the code and perform the action on the specified window or control.

Paste - RoboScripter

This button will paste the buffer code to the Windows Clipboard. To get the contents of the clipboard, launch your editor, and type ctrl-v to paste the code.

Clear - RoboScripter

This button will clear the code buffer completely.

Help - RoboScripter

This button will launch the appropriate help file for the action that is specified.

Options - RoboScripter

This button will allow you to modify RoboScripter settings.

You can check or uncheck the following settings:

- Prompt for program to run on start-up
- Verbose script generation mode
- Show more control choices

Quit - RoboScripter

This button will close the RoboScripter dialog.

Comment / UserInput - RoboScripter

This field displays comments on the action that is selected, or will allow for user input if required.

Title - RoboScripter

This field displays the title of the window / control.

ID - RoboScripter

This field displays the ID of the window / control.