



DesktopX™

V E R S I O N 3

Scripting Guide

Spring 2005

Copyright © 2005 Stardock.net, Inc.

History

Version 3.0, March 8th 2005

- Widget.RegistrerController

Version 3.0, February 28th 2005

- Widget.Preference.Tick

Version 3.0, February 14th 2005

- Object.CurrentFrame
- Widget_OnPreferencesChange callback.
- Password preference type.

Version 3.0, February 11th 2005

- Widget Preferences
- Scriptable Email Notify

Version 3.0, February 9th 2005

- System.CpuActivity
- System.Clipboard
- System.Mute
- System.Volume
- Centralized scripting

Version 2.4, January 17th 2005

- Picture.
- SetPicture.

Version 2.4, January 13th 2005

- Scriptable popup menus.
- Controller objects.

Version 2.4, December 30th 2004

- New layout.
- Added section 1.1 - Namespaces
- Rewritten section 5 - System Callbacks
- Updated System.ScreenWidth, System.VscreenLeft sections
- Rewritten the System.Workarea* chapter
- Updated Sub Object_OnStateChange chapter
- Added "Object.States namespace" chapter
- Corrected information in Object.LocalStorage and Object.PersistStorage chapters
- New Enumerators and Threading chapters.



Contents

1	The Basics of DXScript	7
1.1	INTRODUCTION.....	7
1.2	NAMESPACES	10
1.3	ENUMERATORS.....	11
1.4	THREADING	12
1.5	VBSSCRIPT REFERENCES	13
2	Object namespace Reference.....	14
2.1	OBJECT.STATES NAMESPACE	14
2.2	OBJECT.NAME.....	14
2.3	OBJECT.PARENT	14
2.4	OBJECT.VISIBLE	14
2.5	OBJECT.CLONE.....	16
2.6	OBJECT.DELETE	16
2.7	OBJECT.TOP	16
2.8	OBJECT.BOTTOM.....	16
2.9	OBJECT.LEFT.....	16
2.10	OBJECT.RIGHT	16
2.11	OBJECT.MOVE.....	16
2.12	OBJECT.ROTATION *	16
2.13	OBJECT.HEIGHT	17
2.14	OBJECT.WIDTH	17
2.15	OBJECT.RESIZE	17
2.16	OBJECT.HUE *	17
2.17	OBJECT.BRIGHTNESS *	17
2.18	OBJECT.CONTRAST *	17
2.19	OBJECT.OPACITY *	17
2.20	OBJECT.STATE	18
2.21	OBJECT.STATEPREEMPT	18
2.22	OBJECT.TEXT *	18
2.23	OBJECT.TEXTCOLOR *	18
2.24	OBJECT.TEXTBORDER *	18
2.25	OBJECT.TEXTBORDERCOLOR *	18
2.26	OBJECT.SETTIMER	18



2.27	OBJECT.KILLTIMER	18
2.28	OBJECT.SLEEP.....	18
2.29	OBJECT.EXECUTECOMMAND.....	19
2.30	OBJECT.LOCALSTORAGE	20
2.31	OBJECT.PERSISTSTORAGE	20
2.32	OBJECT.ONTOP	20
2.33	OBJECT.SETFOCUS	21
2.34	OBJECT.TOOLTIPTEXT	21
2.35	OBJECT.APPBAR	22
2.36	OBJECT.DIRECTORY.....	22
2.37	OBJECT.SOUND *	22
2.38	OBJECT.VOLUME	23
2.39	OBJECT.PICTURE.....	23
2.40	OBJECT.SETPICTURE.....	23
2.41	OBJECT.CURRENTFRAME.....	24
2.42	OBJECT.CHILD.....	24
3	Object callbacks.....	25
3.1	OBJECT CALLBACKS AND SCRIPTS	25
3.2	SUB OBJECT_ONSCRIPTENTER.....	25
3.3	SUB OBJECT_ONSCRIPTEXIT.....	26
3.4	SUB OBJECT_ONSTATECHANGE(STATE).....	26
3.5	SUB OBJECT_ONSTATECHANGED(STATE)	26
3.6	SUB OBJECT_ONMOUSEENTER	27
3.7	SUB OBJECT_ONMOUSELEAVE.....	27
3.8	SUB OBJECT_ONSHOW(ISVISIBLE)	27
3.9	SUB OBJECT_ONMOVE(X, Y).....	27
3.10	SUB OBJECT_ONSIZE(WIDTH, HEIGHT)	28
3.11	SUB OBJECT_ONDROPFILES (FILES).....	28
3.12	SUB OBJECT_ONDRAG(X, Y, NEWX, NEWY).....	28
3.13	SUB OBJECT_ONDRAGFINISH	29
3.14	SUB OBJECT_ONSETFOCUS	29
3.15	SUB OBJECT_ONKILLFOCUS	29
3.16	FUNCTION OBJECT_ONCHAR(KEY, EXTENDED).....	30
3.17	FUNCTION OBJECT_ONKEYDOWN(KEY, FLAGS).....	31
3.18	FUNCTION OBJECT_ONKEYUP(KEY, FLAGS).....	31
3.19	FUNCTION OBJECT_ONLBUTTONDOWN(X, Y)	32
3.20	FUNCTION OBJECT_ONRBUTTONDOWN(X, Y).....	32



3.21	FUNCTION OBJECT_ONLBUTTONUP(X, Y, DRAGGED).....	32
3.22	FUNCTION OBJECT_ONRBUTTONUP(X, Y, DRAGGED)	32
4	System namespace	33
4.1	SYSTEM.CURSORSX.....	33
4.2	SYSTEM.CURSORY.....	33
4.3	SYSTEM.PIXELCOLOR	33
4.4	SYSTEM.INTERNETCONNECTED.....	34
4.5	SYSTEM.PING	34
4.6	SYSTEM.SETWALLPAPER.....	34
4.7	SYSTEM.SCREENWIDTH.....	34
4.8	SYSTEM.SCREENHEIGHT.....	34
4.9	SYSTEM.VSCREENLEFT.....	35
4.10	SYSTEM.VSCREENTOP.....	35
4.11	SYSTEM.VSCREENWIDTH.....	35
4.12	SYSTEM.VSCREENHEIGHT.....	35
4.13	SYSTEM.WORKAREALEFT	35
4.14	SYSTEM.WORKAREARIGHT	35
4.15	SYSTEM.WORKAREATOP	35
4.16	SYSTEM.WORKAREABOTTOM	35
4.17	SYSTEM.FOLDERDIALOG (INFO, INITIALDIR, FLAGS).....	35
4.18	SYSTEM.FILEOPENDIALOG (TITLE, DEFAULTFILE, INTIALDIR, EXTENSIONS, FLAGS)	36
4.19	SYSTEM.FILESAVEDIALOG (TITLE, DEFAULTFILE, INTIALDIR, EXTENSIONS, FLAGS).....	36
4.20	SYSTEM.KEYSTATE(VK)	37
4.21	SYSTEM.CLIPBOARD	38
4.22	SYSTEM.CPUACTIVITY	38
4.23	SYSTEM.VOLUME.....	38
4.24	SYSTEM.MUTE	38
5	System Callbacks.....	39
5.1	SUB SYSTEM_ONSCREENCHANGE	39
5.2	SUB SYSTEM_ONWORKAREACHANGE.....	39
6	DesktopX namespace	40
6.1	DESKTOPX.OBJECT.....	40
6.2	DESKTOPX.SCRIPTOBJECT	40
6.3	DESKTOPX.ISOBJECT.....	40
6.4	DESKTOPX.EXECUTABLEDIRECTORY (ONLY FOR PRO APPS).....	40
7	Widget Namespace	41
7.1	WIDGET.MINIMIZE.....	41



7.2	WIDGET.RESTORE.....	41
7.3	WIDGET.ABOUT.....	41
7.4	WIDGET.CLOSE.....	41
7.5	WIDGET.CAPTION.....	41
7.6	WIDGET.AUTORUN.....	41
7.7	WIDGET.REGISTRERCONTROLLER.....	42
8	Widgets preferences.....	43
8.1	WIDGET.ADDPREFERENCE "NAME".....	44
8.2	PREFERENCE.TYPE.....	44
8.3	PREFERENCE.CAPTION.....	44
8.4	PREFERENCE.DEFAULTVALUE.....	44
8.5	PREFERENCE.DESCRPTION.....	44
8.6	PREFERENCE.ADDVALUE.....	44
8.7	PREFERENCE.MINVALUE.....	45
8.8	PREFERENCE.MAXVALUE.....	45
8.9	PREFERENCE.TICKS.....	45
8.10	SUB WIDGET_ONPREFERENCESCHANGE.....	45
9	Scriptable popup menu.....	46
9.1	APPENDMENU.....	46
9.2	TRACKPOPUPMENU.....	47
10	Controller scripts.....	48
11	ActiveX Controls in DXScript.....	50
11.1	PUSHBUTTON CONTROL.....	52
11.2	CHECKBOX CONTROL.....	52
11.3	EDIT CONTROL.....	53
11.4	COMBOBOX CONTROL.....	53
11.5	LISTBOX CONTROL.....	54
12	Scriptable E-Mail Notify plugin.....	56



DXScript

1 The Basics of DXScript

1.1 Introduction

By now you should realize just what amazing results can be achieved using DesktopX. DXScript takes things to an entirely new level!

Whilst DesktopX is flexible, DXScript allows a user with a hint of programming experience to really extend the possibilities of what you can do.

DXScript allows you to program in either VBScript and JScript, two of the most simple and common languages available. This document is not designed to teach you how to program these languages, we assume you know the basics. If not, I suggest you take a little time to learn and then you can join the fun. Before you recoil in fear at the words 'programming' and 'script', this is not a trip into the world of the uber-geek. Let me say this once and say it loud - "Learning DXScript is not hard!".

Adding script to objects is very easy. Simply open up the Object Properties dialogue of the object to which you want to add script and on the General tab you will see a button saying 'New' in the Script section. Clicking this brings up the dialogue you see on the left.

You will note that the main window is prepared for you to start entering script, but before we do that let's look around the 'Script' menu.

Under this there are three sections. The second one, 'Language' allows you to specify whether you want to program in VBScript or JScript. Whilst both work in pretty much the same way, we will be using VBScript examples here.



Once you have decided on a Scripting language you are ready to go. This is where you start to need a bit of scripting or programming knowledge.

As with most programming languages, everything is based around a series of Events and then Methods and Properties are used to make things happen.

By default two events are created. You will also see comment lines (starting with ' ') that explain when they occur. So lets get scripting straight away in the traditional way!

Edit the script to display the following:

```
Sub Object_OnScriptEnter
    MsgBox "Hello, World!"
End Sub
Sub Object_OnScriptExit
    MsgBox "So Long, And Thanks For All The Fish!"
End Sub
```

Note that the indents on the code are just for clarity. If you now close the script you will now have an object the pops up a message every time the script is enabled or disabled. OK, so this is a bit fake because you're enabling the script and disabling script, but in the real world this will pop up a message when DesktopX loads (and the object's script is enabled), and a message when DesktopX unloads (and the object's script exits).

One of the other key Events is `Object_OnStateChange(state)`. This identifies when an object switches from one state to another and allows script to be run at that time. By querying the state that has arisen, script can be run in response to that. In the below example, the script pops up a message if the user activates the object (the 'Command executed' state).

```
Sub Object_OnStateChange(state)
    If state = "Command executed" Then
        MsgBox "You activated me!"
    End If
End Sub
```

Note that you can also check for Custom Messages, and not just the default states that exist. The final state that needs discussing here is the `Object_OnTimer` event.



DesktopX can create timers which mean that script can be run at set intervals. Although we will be discussing object methods later, we need to use one here. Before a timer event can run you need to define that timer by using an `Object.SetTimer` command. In this command you give the timer a unique numeric identifier and specify the interval (in milliseconds) at which the timer will run.

In the below example a timer with the identifier 12345 is set up when the script starts to run every 10 minutes (600000 milliseconds). Each time this timer occurs, the script flashes up the current time using the VBScript command `Time()`.

```
Sub Object_OnScriptEnter
    Object.SetTimer 12345, 600000
End Sub
Sub Object_OnTimer12345
    MsgBox "The time is " & Time()
End Sub
Sub Object_OnScriptExit
    Object.KillTimer 12345
End Sub
```

Now that we have discussed events, obviously you can use any VBScript or JavaScript within an event, but to properly interact with DesktopX you need specific Methods and Properties. More detail and examples are also available in the DXScript Reference, but we will discuss some of the basics here to get you started.

To start with you need to know how to refer to objects and then we'll move onto the things you can do with them.

To refer to the current object (i.e. the one who's script you are editing), simply use `Object.xxxx`. For example, `Object.Left = 200`.

To refer to another object in the theme you use `DesktopX.Object("ObjectName").xxxx`. For example, `DesktopX.Object("Button").Left = 200`.

Note that some objects may have multiple states. Where this occurs, DXScript allows you to adjust parameters for some or all of these states. This is further explained in the Object Namespace reference.



1.2 Namespaces

Namespaces identify a “node” of properties and methods. When you create a new script for an object (“New” button in the General Properties page), you are actually creating an accessible “root” namespace for that object.

Sub-spaces group common functionality for the object. For instance:

`Object.Left:`

Let you access (read/write) the Left property from the Object namespace.

`Object.State(“Mouse over”).Picture`

Let you read/write the Picture property for the State(“Mouse over”) namespace, that is accessed from the Object namespace.

`Object.Parent`

Let you access the root script namespace of the parent object. From here on, you can do as above: `Object.Parent.Object.Left`, `Object.Parent.Object.State(“Mouse over”).Picture`.

Important: It assumes the parent object actually has a script namespace, i.e. it has a script associated. If this isn’t the case, you’ll get a script error when trying to access the parent members.

`DesktopX.ScriptObject(“object01”)`

Let you access the root namespace for object01. This means you can access all sub namespaces like Object and Control, global variables and functions.

`DesktopX.ScriptObject(“object01”).Object.Name`

Name property for the “object01” object.

`DesktopX.ScriptObject(“object01”).myvariable1`

Read/write myvariable1 declared in the “object01” script.

`DesktopX.ScriptObject(“object01”).myfunc1 args`

Call myfunc1 declared in the “object01” script.

`DesktopX.Object(“object01”)`

Gives straight access to the plain Object namespace. This is usually a shortcut and memory saving technique, because it let you access the Object namespace without that “object01” actually has a namespace associated.

The next sections will cover each namespace in detail.



1.3 Enumerators

DXScript has support for these object collections:

- DesktopX.Objects
- DesktopX.GroupObjects("groupname")
- Object.Children

They can be used like this:

- DesktopX.Objects.count – number of objects in the collection
- DesktopX.Objects.item("obj1") – returns the Object namespace of "obj1"
- DesktopX.Objects.item(3) – returns the Object namespace of the third object in the collection

You can also use the VB Sript For Each method as shown in the following examples:

```
msgbox "Total objects:" & DesktopX.Objects.count
msgbox "First object name:" & DesktopX.Objects.item(1).name ' or .left or
.on top etc
msgbox "Test1 object name:" & DesktopX.Objects.item("Test1").name
```

```
For Each elem In DesktopX.Objects
msgbox "First object name:" & elem.name
elem.left = 50
'etc
Next
```

Simply put the returned item is identical to the references returned by DesktopX.Object("name"), NOT DesktopX.ScriptObject("name"). You can always do DesktopX.ScriptObject(elem.name) if you need.

DesktopX.GroupObjects("groupname") and Object.Children collections work in a similar way.



1.4 Threading

Starting version 2.40b[a].002 scripts run by default inside the DesktopX GUI thread. This is because scripts mostly work interactively, affect the object appearance, respond to user input etc. It make sense to do such things synchronously with the DesktopX thread.

If you need some long running scripts to be asynchronous, you can enable the menu option in the Script Editor “Run in separated thread”. For instance, if a script isn’t elaborating (downloading from internet etc) for several seconds, it will not lockup DesktopX until it returned.

There is only a caveat in running scripts in the secondary thread: scripts in one thread cannot access the script namespace of a script in another thread. For instance, you cannot use:

`DesktopX.ScriptObject(“object1”).*` if “object1” is in a different thread. However, you can use the `DesktopX.Object(“object1”).*` shortcut as explained before, because it doesn’t hook into the actual script namespace. Of course you can only use Object properties and methods, not access script variables, functions and other sub-namespaces.



1.5 VBScript references

The best way to learn VBScript is via the various resources available online. Here are a few to help you get started:

Microsoft VBScript Reference

(<http://msdn.microsoft.com/library/en-us/script56/html/vtoriVBScript.asp>)

W3Schools VBScript Tutorial

(<http://www.w3schools.com/vbscript/default.asp>)



2 Object namespace Reference

This section defines the scripting commands that can be applied to objects in DesktopX.

2.1 *Object.States namespace*

From the object namespace you can access the States namespace to read/write state specific properties. States properties also exist in the Object namespace. There are basically three different methods you can use to read/write:

`Object.property` – it'll read/write the state property for the current state

`Object.States("mystate1").property` - it'll read/write the state property for the "mystate1" state

`Object.States("").property` – it'll write the property to all states in the object

Unless the object has only one state, it is usually better to use the second method.

All properties which can be applied in the States namespace as well as in the Object namespace are suffixed with *

2.2 *Object.Name*

2.3 *Object.Parent*

2.4 *Object.Visible*

Through use of these commands you can set an objects name, though it is more likely you will want to retrieve the name of an object or its parent. `Object.Parent` is used to retrieve the entire object model of the parent, so you can refer to the object. In addition to this, you can also set an object's parent using this property.

The 'Visible' property is most useful allowing you to show or hide an object without the use of popups or messaging.

Examples:

1) `MsgBox "This object is called " & Object.Name`



- 2) MsgBox "This object's parent is " & Object.Parent.Object.Name
- 3) Object.Parent = DesktopX.Object("Some object")
- 4) DesktopX.Object("AnObject").Parent = DesktopX.ScriptObject("anotherobject")
- 5) Object.Visible = False



2.5 Object.Clone

2.6 Object.Delete

This allows you to either duplicate or delete the specified object. When duplicating an object you need to provide the name for the new object and the x and y coordinates where it will be placed. Deleting objects may be useful for things like deleting temporary objects such as instructions once the user has taken note of them.

Examples:

```
Object.Clone "myobject", 300, 200  
If MsgBox("Delete object", vbYesNo) = vbYes Then Object.Delete
```

2.7 Object.Top

2.8 Object.Bottom

2.9 Object.Left

2.10 Object.Right

2.11 Object.Move

2.12 Object.Rotation *

With these positioning properties, you can explicitly set or retrieve the location of one side of the object.

If you want to reposition an object, the most efficient way to do it is via 'Move'. To do this you specify x and y coordinates in pixels of where you want to place the object.

You can also rotate an object using Object.Rotation.

Examples:

```
Object.Top = 500  
Object.Move 500,200  
  
For x = 1 To 10  
    Object.Rotation = x*36  
    Object.Sleep 200  
Next
```



2.13 Object.Height

2.14 Object.Width

With this command you can retrieve or redefine the actual height and width of the object. Obviously this will stretch the graphic used until it reaches this size.

Examples:

```
Object.Height = 120  
Object.Width = Object.Width * 2
```

2.15 Object.Resize

With this command you can quickly and easily resize an object in one command rather than setting width and height individually.

Examples:

```
Object.Resize 100,300  
Object.Resize Object.Width + 10, Object.Height + 10
```

2.16 Object.Hue *

2.17 Object.Brightness *

2.18 Object.Contrast *

2.19 Object.Opacity *

If you want to dynamically change the color, hue or brightness of an object, it is easy to do with these commands. Simply set a value of 1-255 for hue, -255 – 255 for brightness and -100 – 100 for contrast. Set it to 0 to remove any previously made changes.

You can also set or retrieve the level of opacity that an object has in a range from 0 (totally invisible) to 100 (totally visible).

Examples:

```
1) For x = 0 To 255  
    Object.Hue = x  
Next  
2) Object.Brightness = -30  
3) DesktopX.Object("anotherobject").Contrast = 40
```



4) `Object.Opacity = 60`

2.20 *Object.State*

2.21 *Object.StatePreempt*

Through use of `Object.State` you can change or retrieve the state of an object. This is one of the most common commands used in script as it is used to trigger objects to act certain ways. `Object.StatePreempt` should be used when you want to set a state immediately without waiting for any animations etc to complete.

Examples:

- 1) `If Object.State = "Command executed" Then ...`
- 2) `x = DesktopX.Object("anotherobject").State`
- 3) `DesktopX.ScriptObject("scr_object").Object.StatePreempt = "Hide"`

2.22 *Object.Text **

2.23 *Object.TextColor **

2.24 *Object.TextBorder **

2.25 *Object.TextBorderColor **

If your object is a text object rather than an image, you can manipulate it using script changing either the text itself or the appearance. You can set/retrieve the text or its color and well as defining whether the text has a border, and if so what color it is.

The colors are most easily set using an RGB value separated via commas.

Examples:

- 1) `If Instr(Object.Text, "Data") Then MsgBox "String found"`
- 2) `Object.TextColor = RGB(120,200,255)`
- 3) `Object.TextBorder = True`
- 4) `Object.TextBorderColor = RGB(255,0,0)`

2.26 *Object.SetTimer*

2.27 *Object.KillTimer*

2.28 *Object.Sleep*

As discussed earlier, the `Object.SetTimer` sets an event and an interval to which you can then add code to run at the predefined interval. If you need to stop a timer running you can use `KillTimer`



to stop it. e.g. `Object.KillTimer 12345` will stop any code in the `Object_OnTimer12345` event from running.

`Object.Sleep` stops code running temporarily for a defined period of time. For example `Object.Sleep 1000` will wait for 1 second before continuing to run.

Example:

```
Sub Object_OnScriptEnter
    Object.SetTimer 12345, 600000
End Sub
Sub Object_OnTimer12345
    MsgBox "The time is " & Time()
End Sub
Sub Object_OnScriptExit
    Object.KillTimer 12345
End Sub
```

2.29 *Object.ExecuteCommand*

This executed any command associated with the object in it's Object Type. This is particularly useful for doing things at set intervals or events, such as when DesktopX exits.

Example:

```
Sub Object_OnScriptExit
    Object.ExecuteCommand
End Sub
```



2.30 Object.LocalStorage

2.31 Object.PersistStorage

When coding it is often useful to store persistent information which can be retrieved and used as required across multiple executions of the same object or widget.

To store data you need to give the data a unique reference (for that object) and set it's value. For example `Object.LocalStorage("MyZip") = 48152` would place the value 48152 in a storage variable called MyZip. `"MyZip"=48152` will be automatically saved and restored when the object is unloaded and reloaded.

The difference between the two types is in its persistence across object packaging and distribution. LocalStorage will NOT be saved when the object is saved as .dxdpack or a widget is built. PersistStorage instead will save its value.

LocalStorage is useful to store personal information, like a password or a ZIP code. Infact, you don't want such information to be preserved when you export and redistribute the object to other people. However, you want these values to be preserved across multiple run of the same object/widget.

Example:

```
Sub Object_OnScriptEnter
    If Object.LocalStorage("MyZip") = "" Then
        Object.LocalStorage("MyZip") = "48152"
    End If
    Object.SetTimer12346, 3600000
End Sub
Sub Object_OnTimer12346
    GetWeather(Object.LocalStorage("MyZip"))
End Sub
Sub Object_OnScriptExit
    Object.KillTimer 12346
End Sub
Function GetWeather(zip)
    ...
End Function
```

2.32 Object.OnTop

The object pushes an object to the top of the z-order which obviously makes it more visible. The below example makes an object appear above other objects when you move the mouse over it.



Example:

```
Sub Object_OnStateChange(state)
    If state = "Mouse over" Then
        Object.OnTop
    End If
End Sub
```

2.33 Object.SetFocus

This simply sets an object's focus so it can respond to events. For example, all objects have an `Object_OnSetFocus` so this will be triggered if this command is used. Also, where a text-based DesktopX object responds to functions based on keyboard or mouse activity (e.g. Function `Object_OnChar(dwKeyCode, flag)`, Function `Object_OnLButtonDown(x, y)`) then it will respond when these events occur. You can also apply this to ActiveX controls, so for example if an object contains a DesktopX Edit Control then setting its focus will prepare it to accept text input.

Example:

```
Sub Object_OnStateChange(state)
    If state = "Command executed" Then
        MsgBox "Ready for input"
        Object.SetFocus
    End If
End Sub
```

2.34 Object.TooltipText

This allows you to set the tooltip of the object which is particularly useful to provide additional information to the user. You can also use this to provide different information depending on different circumstances.

Example:

```
If Object.Text = "New mail" Then
    Object.TooltipText = "Click to launch mail software"
Else
    Object.TooltipText = " "
End If
```



2.35 *Object.AppBar*

An AppBar is an object that is designed to be attached to the edge of the screen like the Taskbar. Also like the taskbar it can be set to autohide, but beyond this you can undock it as well so it can be moved on the screen. When an appbar is set to Autohide, then moving the mouse over the edge of the screen will cause the AppBar to smoothly appear.

Note that `Object.AppBar` can only be written, not read, so if you need to check the mode at any time you need to set a variable when you set the mode, and then query the value of this variable. Example 1 shows how you may set a variable in this manner.

The values for `Object.AppBar` are as follows:

- 0 = Disabled
- 1 = Docked
- 2 - Autohide

Example:

```
1) Sub Object_OnScriptEnter
    Object.AppbarMode = 1
    appmode = 1
End Sub

2) If state = "Command executed" Then
    DesktopX.Object("mainedock").AppbarMode = 2
End If
```

2.36 *Object.Directory*

This tells you the directory within which the object is located. This will point to the user's theme directory.

2.37 *Object.Sound* *

This allows you to set the sound associated with the object either globally or specific to certain states. The sound file targeted can either be a WAV or MP3 format file.

Example:

```
1) Object.Sound = "c:\mydirectory\mytune.mp3"
2) Object.State("Mouse down").Sound = "ding.wav"
```



2.38 *Object.Volume*

This allows you to set the volume of the sound played by an object. The range is 0 (muted) to 100 (full system volume)

Example:

- 1) `Object.Volume = 80`
- 2) `Object.Volume = Object.Volume + 10`

2.39 *Object.Picture*

This allows you to get/set the picture of an object or state.

Example:

- 1) `Object.Picture = "image01.png"`
- 2) `Object.States("Mouse away").Picture = "C:\images\image01.png"`
- 3) `Object.Picture = http://www.something.com/pic.png`

Notes:

- You can use the following modes:
 - o File names: DX will check into the current theme or widget folder. Note that files must be registered as custom files or bound to at least one state for them to be packed into a .dxtheme, .dpack or .exe.
 - o Full path: this can be useful for totally dynamic things like a Picture viewer widget in that you can simply do:

```
Sub Object_OnDropFiles(files)
Object.picture = files
End Sub
```

 - Full path images are not exported.
 - o Remote paths: you can use this to easily make a webcam object.
 - Remote path files are not exported.

2.40 *Object.SetPicture*

This method let you set the picture of an object or state AND its other properties in one call.

Syntax:

```
Object.SetPicture fileName, frames, interval, flags
```

For fileName see `Object.Picture`.

Frames is the number of frames in the picture.

Interval is the number of milliseconds between each frame.

Flags is a combination of the following flags:



&H00000001 – Loop

&H00000002 – Reverse

&H00000004 – Alternate

&H00000008 – Interruptable

&H00000010 – Static

2.41 *Object.CurrentFrame*

Gets/sets the current frame of the animation. In order to use this, the animation should be set “Scripted” in the Properties panel.

2.42 *Object.Child*

Returns true if the object is a contained child. If it has a parent but it is only “owned” (Child = No in Summary page) , it will return false.

Contained children coordinates are relative to the parent’s top/left corner.



3 Object call-backs

These are events to which an object can respond. Script can be placed within these Subroutines and Functions to perform actions when these events occur.

For functions, you should also return True or False at the end of the function as a clean coding practice. Returning True stops DesktopX processing additional events, which means that you can use functions like `Object_OnRButtonDown` to perform actions other than displaying the DesktopX right click menu (if enabled). You should return False otherwise.

3.1 Object callbacks and scripts

Starting 3.0 release, you don't need a script just to respond to an Object callback. Child objects events will be automatically notified to the parent script, if one exists.

It means you only need one script in the root parent object to get all events from its descendants, for instance `Object_OnLButtonUp`.

To support this new style of coding and support pre-3.0 scripts at the same time, all callbacks described in this chapter, except for `Object_OnScriptEnter` and `Object_OnScriptExit`, also exist in *Ex form. *Ex callbacks have an additional first parameter of type Object.

Examples:

Standard callback:

```
Sub Object_OnLButtonDown(x,y) 'only called if the object has a script associated
```

Ex callback:

```
Sub Object_OnLButtonDownEx(obj,x,y) 'receives events from the object and all its children
```

```
Select case obj.name
```

```
    Case "mybutton01"
```

```
        'do something
```

```
    Case "mybutton02"
```

```
        'etc
```

```
End Select
```

```
End Sub
```

3.2 Sub Object_OnScriptEnter

Occurs as soon as the script is enabled, which usually occurs when an object is loaded.



Example:

```
Sub Object_OnScriptEnter
    Object.Text = Object.PersistStorage("mytext")
End Sub
```

3.3 Sub Object_OnScriptExit

Occurs as soon as the script is disabled, which usually occurs when an object is unloaded.

Example:

```
Sub Object_OnScriptExit
    Object.PersistStorage("mytext") = Object.Text
End Sub
```

3.4 Sub Object_OnStateChange(state)

3.5 Sub Object_OnStateChanged(state)

When the state of the object changes state these events are called and the variable (state) is identified allowing this one event to deal with all states.

The differentiating factor is that OnStateChange is called as the change commences, and the OnStateChanged event occurs when the state change has completed. OnStateChanged is particularly useful for the synchronization of animated objects and effects.

Note: Setting Object.StatePreempt inside Object_OnStateChange will actually have the special effect of switching the state being changed to the specified state. For instance you can hijack "Mouse over" messages to "MyMover1" and "MyMover2" depending on some state information.

Note: You can usually more effectively use Object_OnMouseEnter and Object_OnMouseLeave notifications instead of checking for "Mouse over" and "Mouse away" states in Object_OnStateChange. It is more efficient because those events are direct and don't rely on animation delays and queued animated states completions.

Note: Any references to states should be case sensitive.

Example:

```
Sub Object_OnStateChange(state)
```



```
    If state = "Mouse over" Then
        Object.Opacity = 100
    ElseIf state = "Mouse away" Then
        Object.Opacity = 50
    End If
End Sub
```

3.6 Sub Object_OnMouseEnter

3.7 Sub Object_OnMouseLeave

This is the cleaner way to check for user mouse interaction with an object. By using this you can avoid your code for these events being combined with other the state change code. You can use the OnMouseButton functions described later in combination with these very effectively.

Example:

```
Sub Object_OnMouseEnter
    Object.Opacity = 100
End Sub
Sub Object_OnMouseLeave
    Object.Opacity = 50
End Sub
```

3.8 Sub Object_OnShow(IsVisible)

This function is triggered whenever the visibility of an object changes. A single variable is respond which is “True” or “False” depending on whether the object is being shown or hidden.

Example:

```
Sub Object_OnShow(IsVisible)
    If IsVisible = True Then
        msgbox "Showing object"
    Else
        msgbox "Hiding object"
    End If
End Sub
```

3.9 Sub Object_OnMove(x, y)

This function is triggered whenever the position of an object changes, but it via mouse or keyboard movement, or by script manipulation. The coordinates of it’s new position are returned.

Example:

```
Sub Object_OnMove(x,y)
    If x < 100 Then Object.Left = 100
```



End Sub

3.10 Sub Object_OnSize(width, height)

If the objects size is adjusted then you can react to this event using this subroutine. In the following example the event ensures that the proportions of the object are constrained if the object gets resized.

Example:

```
Sub Object_OnSize(width, height)
    Object.Height = Object.Width / 2
End Sub
```

3.11 Sub Object_OnDropFiles (files)

This event is triggered if the user drags one or more files onto the object and releases the mouse. A variable is returned containing the full path of all the files separates by a pipe (“|”) character.

Example:

```
Dim filelist
Sub Object_OnDropFiles(files)
    filelist = Split(files,"|")
    For x = 0 To UBound(filelist)
        outputmsg = outputmsg & filelist(x) & vbNewLine
    Next
    msgbox outputmsg
End Sub
```

3.12 Sub Object_OnDrag(x, y, newX, newY)

This event is fired as the object is dragged. The x and the y coordinates correspond the where on the object the click occurred and the “newPos” coordinated specify the top left position of the object in the position it has been dragged to.

If the object’s position is locked then the x,y coordinated report a position relative to where the object was originally clicked.

You can get the position of the object before it was dragged using Object.Left and Object.Top

The example below allows you to drag an object to within 100 pixels of the primary monitor screen edge but no further.

Example:

```
Sub Object_OnDrag(x, y, newX, newY)
```



```
If newX < 100 Then Object.Left = 100
If (newX + Object.Width) > System.ScreenWidth - 100 Then
    Object.Right = System.ScreenWidth - 100
End If
If newY < 100 Then Object.Top = 100
If (newY + Object.Width) > System.ScreenHeight - 100 Then
    Object.Bottom = System.ScreenHeight - 100
End If
End Sub
```

3.13 Sub Object_OnDragFinish

This event occurs when you finish dragging the object so you can react to the new position of the object. For example, the script below ensures that after an object has been moved then a second object is placed directly underneath it wherever it is placed.

Example:

```
Sub Object_OnDragFinish
    DesktopX.Object("obj2").Top = Object.Bottom
    DesktopX.Object("obj2").Left = Object.Left
End Sub
```

3.14 Sub Object_OnSetFocus

3.15 Sub Object_OnKillFocus

These events occur when an object receives or loses the focus. This means that you can react to a user starting to interact with or ending interaction with an object. You may just want to draw attention to the fact that the object has the focus or do something more like validate the input of a DesktopX Edit control if the user tries to leave it.

Example:

```
Sub Object_OnSetFocus
    Object.state = "FocusON"
End Sub

Sub Object_OnKillFocus
    Object.state = "FocusOFF"
End Sub
```



3.16 Function *Object_OnChar(key, extended)*

If an object has the focus then this function is called when a key is depressed and the ASCII character code is returned in the variable. Note that 'a' and 'A' return different values so this event is well suited to responding to a user typing. It also returns a code to represent extended variables. These are not really necessary to interpret and can be ignored.

Example:

```
Function Object_OnChar(key, extended)
    MsgBox "You pressed the " & Asc(key) " key which has the ASCII value of
    " & key
    Object_OnChar = False
End Sub
```



3.17 Function *Object_OnKeyDown(key, flags)*

3.18 Function *Object_OnKeyUp(key, flags)*

This returns the actual key pressed rather than the ASCII value of the character returned. As such it is better suited to when you want to return the actual key such as an arrow key or Shift key. Note that in Edit mode certain keys such as the arrow key will move the object rather than respond to your code, but when in User mode as you should be whenever possible it will work fine.

You can get a list of the valid key values here:

<http://msdn.microsoft.com/library/en-us/winui/winui/windowsuserinterface/userinput/virtualkeycodes.asp>

You need to define the constant at the beginning of the script if you want to use a textual name for clarity.

There is only really one useful extended value which will stop a character from repeating. This is shown in the second example.

Example:

```
Const VK_SHIFT = &H10 'Shift Key
Function Object_OnKeyDown(key, extended)
    If key = VK_SHIFT Then
        MsgBox "Shift pressed"
    End If
    Object_OnKeyDown = False
End Function
```

The below example will move an object when it is selected and the enter key is pressed, but will not repeat the movement if the key is help; so the user must actively click the key again to do this.

```
Const EnterKey = &H0D
Const Repeat = &H40000000
Function Object_OnKeyDown(key, flags)
    If key = EnterKey Then
        If Repeat <> (flags And Repeat) Then
            Object.Move Object.Left + 10, Object.Top
        End If
    End If
End Function
```



3.19 Function *Object_OnLButtonDown(x, y)*

3.20 Function *Object_OnRButtonDown(x, y)*

3.21 Function *Object_OnLButtonUp(x, y, Dragged)*

3.22 Function *Object_OnRButtonUp(x, y, Dragged)*

These functions are called as soon as the corresponding mouse button is pressed or released. In all cases two variables are returned which are the x and y coordinates within the object (i.e. not the screen position). In the ButtonUp functions, a third is returned True or False depending on whether the object has been dragged or not.

Example:

```
Function Object_OnLButtonDown(x, y)
    Object.PersistStorage("x") = Object.Left
    Object.PersistStorage("y") = Object.Top
    Object_OnLButtonDown = False
End Function
```

```
Function Object_OnLButtonUp(x, y, Dragged)
    If Dragged = True Then
        MsgBox "You moved the object " & Object.Left - Object.PersistStorage("x")
        & " pixels horizontally and " _
        & Object.Top - Object.PersistStorage("y") & " pixels vertically"
    End If
    Object_OnLButtonUp = False
End Function
```



4 System namespace

Another set of properties can be retrieved about the actual system on which DesktopX is running. This allows you to retrieve information and perform actions dependant on the value.

4.1 *System.CursorX*

4.2 *System.CursorY*

These returns the current coordinates of the mouse cursor.

Example:

```
1) Object.Text = "X:" & System.CursorX & " Y:" & System.CursorY
```

4.3 *System.PixelColor*

This returns the color of the pixel at the specified coordinates.

Example:

```
Sub Object_OnTimer1
    hexcolor = Hex(System.PixelColor(System.CursorX, System.CursorY))
    red = Right(hexcolor, 2)
    green = Mid(hexcolor, 2,2)
    blue = Left(hexcolor, 2)
    Object.Text = CStr(CLng("&H" & red)) & ", " & CStr(CLng("&H" & green)) &
    ", " & CStr(CLng("&H" & blue))
End Sub
```



4.4 System.InternetConnected

4.5 System.Ping

Many good examples of DXScript objects make use of the Internet, so it makes sense to detect whether access to the Internet is available. You can also check the speed of access to a web address (ping) by using System.Ping.

Examples:

- 1) If System.InternetConnected = False Then MsgBox "Go online"
- 2) x = System.Ping("www.wincustomize.com")

4.6 System.SetWallpaper

This allows you to specify the Windows wallpaper. You need to provide a full path to the wallpaper and then an option to define how to display the wallpaper.

Option:

- 0 = use default wallpaper
- 1 = Center wallpaper
- 2 = Tile wallpaper
- 3 = Stretch wallpaper

Examples:

```
System.SetWallpaper "C:\temp\wall1.bmp", 1
System.SetWallpaper Object.Directory & "wall1.jpg", 1
System.SetWallpaper "", 0      This will restore the original wallpaper.
```

4.7 System.ScreenWidth

4.8 System.ScreenHeight

System.ScreenHeight and System.ScreenWidth return the height and width of the primary monitor. This is mostly kept for backward compatibility. New objects should use System.Vscreen* properties instead, since these are multimonitor compatible (see the next chapter).

Examples:

```
Msgbox "Primary monitor resolution: " & System.ScreenHeight & "x" &
System.ScreenWidth
```



4.9 System.VscreenLeft

4.10 System.VScreenTop

4.11 System.VScreenWidth

4.12 System.VScreenHeight

These are the suggested properties to use when checking the monitor coordinates, since they take in consideration the whole virtual screen area, and not only the primary monitor. This will still work in a single monitor setup but will also support multi-monitor setups.

Note that the virtual screen origin is NOT generally (0,0), but it is (VscreenLeft, VscreenTop). Because of this .VscreenWidth returns the WIDTH of the virtual screen, not the right border! To calculate the right and bottom borders use the following code:

```
VirtualScreenRight = System.VScreenLeft + System.VScreenWidth  
VirtualScreenBottom = System.VScreenTop + System.VScreenHeight
```

4.13 System.WorkareaLeft

4.14 System.WorkareaRight

4.15 System.WorkareaTop

4.16 System.WorkareaBottom

These properties retrieve the boundaries of the desktop workarea. This is usually even better than System.Vscreen* properties to base calculations for moving and keeping objects in the wanted position, since it represents the actual “available” working area.

4.17 System.FolderDialog (info, initialDir, flags)

This shows a dialog that allows the user to select a specific folder which can then be acted upon. The parameters to provide are a string which places information at the top of the dialog, the path where browsing should commence from, and then a range of customization flags. The full list of the flags is shown below:

```
'Only file system directories  
BIF_RETURNONLYFSDIRS = &H1  
'No network folders below domain level  
BIF_DONTGOBELOWDOMAIN = &H2  
'Allows user to rename selection  
BIF_EDITBOX = &H10
```



```
'Insist on valid edit box result (or CANCEL)
BIF_VALIDATE = &H20
'Allow URLs To be displayed Or entered
BIF_BROWSEINCLUDEURLS = &H80
'Only returns computers
BIF_BROWSEFORCOMPUTER = &H1000
'Only returns printers
BIF_BROWSEFORPRINTER = &H2000
'Browse for everything
BIF_BROWSEINCLUDEFILES = &H4000
'Sharable resources displayed
BIF_SHAREABLE = &H8000
```

Examples:

```
Const BIF_RETURNONLYFSDIRS = &H1
x = System.FolderDialog ("Please select your image folder:", "c:\",
BIF_RETURNONLYFSDIRS)
Const BIF_EDITBOX = &H10
x = System.FolderDialog ("", "", BIF_EDITBOX)
Const BIF_BROWSEINCLUDEFILES = &H4000
x = System.FolderDialog ("Please select the directory:",
DesktopX.ExecutableDirectory, BIF_BROWSEINCLUDEFILES)
```

4.18 System.FileOpenDialog (title, defaultFile, intialDir, extensions, flags)

4.19 System.FileSaveDialog (title, defaultFile, intialDir, extensions, flags)

This shows a dialog that allows the user to select a specific folder which can then be acted upon. The parameters are a title for the dialog, the default file name, the path where browsing should commence from. You then specific the file types to select in the dialog in a series of pipe (|) separated description/extention pairs such as Text files|*.txt|All files|*.* .

Finally, you can provide a range of customization flags. The full list of the flags is shown below:

```
'Causes the Read Only check box to be selected when the dialog is created
OFN_READONLY = &H1
'Causes the Save As dialog box to generate a message box if the selected file
already exists. The user must confirm whether to overwrite the file
OFN_OVERWRITEPROMPT = &H2
'Hides the read only check box
OFN_HIDEREADONLY = &H4
'Restores the current directory if the user changes directory while searching
```



OFN_NOCHANGEDIR = &H8

'Allow invalid characters in the returned file name

OFN_NOVALIDATE = &H100

'Specifies that the user typed a file name extension that differs from the default extension specified

OFN_EXTENSIONDIFFERENT = &H400

'Specifies that the user can type only valid paths and file names

OFN_PATHMUSTEXIST = &H800

'User can type only names of existing files in the File Name entry field

OFN_FILEMUSTEXIST = &H1000

'If a file that doesn't exist is selected ask for confirmation to create it

OFN_CREATEPROMPT = &H2000

'If the command fails because of a network sharing violation, error is ignored

OFN_SHAREAWARE = &H4000

'Specifies that the returned file does not have the Read Only check box selected and is not in a write-protected directory

OFN_NOREADONLYRETURN = &H8000

'Specifies that the file is not created before the dialog box is closed

OFN_NOTESTFILECREATE = &H10000

'Hides and disables the Network button

OFN_NONETWORKBUTTON = &H20000

'Directs the dialog box to return the path and file name of the selected shortcut (.LNK) file. If this value is not specified, the dialog box returns the path and file name of the file referenced by the shortcut

OFN_NODEREFERENCELINKS = &H100000

Examples:

```
x = System.FileSaveDialog("My title", "new.txt", "c:\", "Text files|*.txt|All files|*.*",0)
```

```
x = System.FileOpenDialog("Select document ...", "", Desktopx.ExecutableDirectory & "docs", "DesktopX Help|*.pdf", 0)
```

4.20 System.KeyState(vk)

This allows to establish the state of any given key from the list available here:

<http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/UserInput/VirtualKeyCodes.asp>

The value returned is either 0 (), 1 (), or 2 () depending on the key state.

Example:

```
Function Object_OnLButtonUp(x,y,dragged)
```



```
Const VK_NUMLOCK = &H90
x = System.KeyState(VK_NUMLOCK)
If x = 0 Then
    MsgBox "NumLock is Off"
ElseIf x = 1 Then
    MsgBox "NumLock is being pressed"
ElseIf x = 2 Then
    MsgBox "NumLock is On"
End If
End Function
```

4.21 System.Clipboard

Property to get and set the text content of the system clipboard.

Example:

```
System.Clipboard = Object.text
```

4.22 System.CPUActivity

Property to get the percentage of current CPU activity.

Example:

```
Object.text = "CPU usage: " & System.CPUActivity & "%"
```

4.23 System.Volume

Property to get/set the system audio volume level (0-100).

4.24 System.Mute

Property to get/set the system audio muting (boolean).



5 System Callbacks

5.1 *Sub System_OnScreenChange*

If the screen resolution changes then you may wish to resize or reposition the objects accordingly.

Since this is supposed to be a custom repositioning code, depending of particular wanted behaviours, it is recommended that you set the automatic repositioning in the “Relation” tab to “Disabled” for both horizontal and vertical axis.

Example:

```
Sub System_OnScreenChange
    Object.Width = System.ScreenWidth - 350
End Sub
```

5.2 *Sub System_OnWorkareaChange*

If the workarea changes (as result of screen resolution changes or taskbar resize) you may want to resize or reposition the objects accordingly.

Since this is supposed to be a custom repositioning code, depending of particular wanted behaviours, it is recommended that you set the automatic repositioning in the “Relation” tab to “Disabled” for both horizontal and vertical axis.

Example:

```
Sub System_OnScreenChange
    Object.Width = System.ScreenWidth - 350
End Sub
```



6 DesktopX namespace

These allow you to refer to the DesktopX environment, the objects within in and also the DesktopX application itself.

6.1 *DesktopX.Object*

6.2 *DesktopX.ScriptObject*

You use these two items to refer to objects other than the current one. You should use the first for non scripted objects.

Examples:

```
DesktopX.Object("mytextobject").Text = "Hello"  
DesktopX.Object("otherobject").Top = Object.Bottom
```

If the object is scripted however, you need to identify whether you are referring to the object itself or the control contained within.

Examples:

```
DesktopX.ScriptObject("scr_textobject").Object.Text = "Hello"  
DesktopX.ScriptObject("scr_textcontrol").Object.Left = 300  
DesktopX.ScriptObject("scr_textcontrol").Control.Text = "Type"
```

6.3 *DesktopX.IsObject*

This allows you to query the existence of another object in the theme which you may choose to do before you interact with it, because it could have been deleted.

Example:

```
Sub Object_OnScriptEnter  
    If DesktopX.IsObject("textobj") = False Then  
        MsgBox "Sorry - important data is missing!"  
    End If  
End Sub
```

6.4 *DesktopX.ExecutableDirectory (only for Pro apps)*

If you are running DesktopX Pro, then the created EXE may be running from any location, so it is sometimes useful to know where this is. DesktopX.ExecutableDirectory will return this directory.



7 Widget Namespace

If you are creating a widget there are several script commands that you can use to replicate the functionality of the widgets and provide additional user feedback.

7.1 *Widget.Minimize*

7.2 *Widget.Restore*

7.3 *Widget.About*

7.4 *Widget.Close*

These simply replicate the options available from the various widget menus to allow you to achieve the functionality via script.returns the current coordinates of the mouse cursor.

7.5 *Widget.Caption*

If you have specified that your widget is displayed in the taskbar then this command will set the text of that taskbar item. Where a system tray icon is used, it will set a tooltip for that item.

Example:

```
1)Widget.Caption = "This has been updated"
```

7.6 *Widget.Autorun*

A user can specify whether to run an application on startup by right clicking the object, but you can also access this functionality via script. This allows you to either prompt the user when they first run the object, or to provide some sort of menu option.

Example:

```
If Widget.Autorun = False Then
x = MsgBox ("Would you like to run this object when you start Windows?",
vbYesNo + vbQuestion, "Autorun ...")
If x = vbYes Then
    Widget.Autorun = True
End If
End If
```



7.7 *Widget.RegistrerController*

Normally object events are fired to the parent if the child isn't handling them directly. If the parent isn't handling them, events are lost. You can configure a script of a widget to be the target of all unhandled events, regardless of parent/children relations.

To do that you can write the following code in `Object_OnScriptEnter`

```
Widget.RegistrerController Object.name
```



8 Widgets preferences

DesktopX 3.0 widgets and gadgets can use an integrated mechanism to manage user preferences.

A “preference” is a local per-user setting that is initially set to a default value and then maintained across multiple executions of the same widget for the same user.

Widget preferences can integrate into the standard widget properties dialog, or can be kept hidden if the author only likes to use persistency and not the interactive feature.

Preferences should be configured in script in `Object_OnScriptEnter`.

To add a preference object use:

```
Widget.AddPreference "PreferenceName"
```

To access a preference object use:

```
Widget.Preference("PreferenceName")
```

To access a preference property use:

```
Widget.Preference("PreferenceName").property
```

Example:

```
Widget.AddPreference "ZIPCode"  
Widget.Preference("ZIPCode").Type = "Text"  
Widget.Preference("ZIPCode").Caption = "ZIP Code"  
Widget.Preference("ZIPCode ").DefaultValue = "12345"  
Widget.Preference("ZIPCode").Description = "It defines the ZIP area code to be  
used by the weather widget"
```

It creates a ZIPCode item of Text type and is initially set to “12345” with the given description for the Properties panel.

To access the configured value you can later use:

```
Widget.Preference("ZIPCode ").Value
```

When the user changes the preferences through the Widget Properties dialog, the script that configured the preferences receives the following event:

```
Sub Widget_OnPreferencesChange()
```



Here is the complete Preference namespace reference:

8.1 *Widget.AddPreference "name"*

Adds a new preference item. The name should not contain spaces. Use `Preference.Caption` to assign a pretty name.

8.2 *Preference.Type*

String property. It defines Valid types are:

“Hidden”: No control will be displayed in the Widget properties panel. Hidden is the default type.

“Text”: Text box control type.

“Password”: Text box with obfuscated characters for password entries.

“Checkbox”: Checkbox control. Values for checkboxes are “1” or “0”.

“ComboEdit”: Editable text combobox control.

“ComboList”: Dropdown list combobox control.

“Slider”: Slider control. Values for sliders are still in string form.

8.3 *Preference.Caption*

Sets the pretty name used to name the preference in the Properties panel.

8.4 *Preference.DefaultValue*

Use to set the preference default value.

8.5 *Preference.Description*

Use to set the preference description that appears in the Widget properties panel.

8.6 *Preference.AddValue*

Use to add an item to a Combobox preference.

Example:

```
Widget.Preference("MyCombo1").AddValue "Green"
```

```
Widget.Preference("MyCombo1").AddValue "Red"
```



8.7 Preference.MinValue

8.8 Preference.MaxValue

Use to configure the minimum and maximum values of a slider preference.

Example:

```
Widget.Preference("Slidersample").MinValue = 1  
Widget.Preference("Slidersample").MaxValue = 5
```

8.9 Preference.Ticks

It specifies the ticks distance for slider preferences.

Example:

```
Widget.Preference("Slidersample").MinValue = 10 'page size of 10 and displayed  
tick each 10 values.
```

8.10 Sub Widget_OnPreferencesChange

Sent when the user applies the preferences in the Widget properties panel.



9 Scriptable popup menu

You can create simple popup menus and replace the default system tray menus for DesktopX and widgets with scripting.

To create a popup menu use DesktopX.CreatePopupMenu.

I.e.

```
Dim menu
```

```
Set menu = DesktopX.CreatePopupMenu
```

```
menu.AppendMenu 0, 1, "Item1"
```

```
menu.AppendMenu 0, 2, "Item2"
```

```
Dim result
```

```
res = m.TrackPopupMenu(0, System.CursorX, System.CursorY)
```

Note: popup menus cannot be used by scripts that run in a separated thread.

Syntax as follow:

9.1 *AppendMenu*

AppendMenu flags, ID, text

ID is the item identifier. When the menu is shown and the user selects an item, TrackPopupMenu will return the ID of the item.

Text is simply the text.

Flags let you specify the following options you can combine:

&H00000800 – Menu separator

&H00000001 – Grayed

&H00000002 – Disabled

&H00000008 – Checked

&H00000010 – Popup

&H00000020 – Menu bar break

&H00000040 – Menu break

&H00000080 – Hilite

Popup (&H00000010) let you add a submenu. You can do like this:

```
Dim submenu
```



```
Set submenu = DesktopX.CreatePopupMenu
submenu.AppendMenu 0, 1, "Item in submenu"
Dim menu
Set menu = DesktopX.CreatePopupMenu
menu.AppendMenu &H00000010, submenu.MenuID, "Sub menu"
menu.AppendMenu 0, 2, "Item in main menu"
```

9.2 TrackPopupMenu

```
Result = TrackPopupMenu(flags, posx, posy)
```

It let you open the menu. It returns the ID of the selected item.

Posx and posy are the screen coordinates where the menu should open.

Flags let you specify some options:

&H00000004 – Center align

&H00000008 – Right align

&H00000010 – Vertical center align

&H00000020 – Bottom align

Example:

```
res = m0.TrackPopupMenu(0, System.CursorX, System.CursorY)
```



10 Controller scripts

In the newest DesktopX release you can register an object and its script to act as a controller for certain functions. Only the controller receives some messages like system tray messages (i.e. the right-click message on the widget right click menu).

To register an object as a controller use `Desktopx.RegisterController ObjectName`

Example:

```
Sub Object_OnScriptEnter
  desktopx.RegisterController Object.Name
End Sub
```

The object will then be called from the `OnControl` notification:

```
Function OnControl(IType, ICode)
```

```
End function
```

The host will call `OnControl` with messages information in `IType` and `ICode`. These messages are available:

IType	ICode	Description
1	1	The systray menu is requested.
1	2	The widget menu is requested from object right-click.
2	0x0201	Left button down
	0x0202	Left button up
	0x0203	Left buttondouble-click
	0x0204	Right button down
	0x0205	Right button up
	0x0206	Right button double-click



Returning true to received messages will cause DesktopX not processing them. For instance you can replace the right click or systray widget menu by returning true to 1 / 1 notification and provide your own menu as described in the preceding chapter.



11 ActiveX Controls in DXScript

The power of script is enhanced by the ability to display ActiveX controls on the desktop instead of using images or text. You can then manipulate these using scripts.

For example, imagine an Excel spreadsheet, Windows Media Player, Combo box, Web Browser all embedded on your desktop. All are possible and very easy to create.

In the DesktopX Script Editor, the first option on the script menu is 'ActiveX Control' from which there is a 'Select Control' option.

We'll look at how to create one of these controls now.

From the list that appears when you click 'Select Control', choose a control and then click OK. In our examples below we will use the 'Microsoft Web Browser' control.

As you will see below, the Script Editor immediately changes to reflect your choice. On the right of the editor you will see a preview of the object, and below that you will see the properties of that object.

Note that the ActiveX Control menu item now has a 'Properties' option where you can toggle to display of this extra information.

If you close the editor via the close button top right or via 'File ... Exit' you will now see a control on your desktop.

You will see that as you move your mouse over the object the cursor changes. You can move the control or resize it using the black handles at the edge.

If you right click the object, you will notice a new item, 'Script Enabled' at the bottom of the list. This is the place where you can toggle the script between a 'running' and 'not running' state. At the moment it is not running which is why the resize handles are visible. Click 'Script Enabled' to toggle the running state and you will see these disappear.

OK, so at the moment it's a dull white box, not very exciting, but nevertheless it's a control and we're about to ramp up the excitement. Let's go back to the editor.

If you right click the control you will note that nothing happens. This is because you are effectively right clicking the Web Browser control, not a DesktopX object. Hold down CTRL so the resize edges appear and right click. You will see that the script is enabled because of the check mark next to that item. Click the Edit Script item and the dialogue will reappear.



OK, try changing the script you see to the one below:

```
Sub Object_OnScriptEnter
    Control.Navigate2 "www.wincustomize.com"
End Sub
```

OK, so exit out of here and enabled the script. Caboom - all of a sudden you have a genuine webpage on your desktop!

You will note from this that scripting is basically the same for Controls as it is for regular objects. The only real difference is that when referring to the ActiveX control you use the Control prefix. Note that you can also use Object methods and properties in the script but these refer to the host object. For example Object.Visible in an object containing an ActiveX control would hide that object and hence that control.

Each ActiveX control will expose it's own properties and these are displayed in the autocomplete feature. This makes it really easy to start coding for a control.

In addition to this it is useful to know which events are associated with a control. With this, we are able to react to events occurring in the control. DesktopX provides a tool to do just this. In the Script Editor, go to the Script Menu and select 'Event Wizard'.

Select the event you want to add and click OK to add it to your script. Click 'Done' when you have added all the events you want to.

You can now add script to that event. For example:

```
Sub Control_DocumentComplete(pDisp, URL)
    MsgBox "Finished downloading the page " & URL
End Sub
```

Now, you will be advised when the page is completely loaded.

One final thing you should know is that DesktopX installs three controls by default. There is an Edit control (to allow you to accept and react to user input), a Push Button control (which acts just like a regular button), and a Check Box control (to give you an on/off switch).

Each of these controls have specific events and properties associated with them.



11.1 *PushButton Control*

The PushButton control allows you to use a regular push button for interaction with your scripts.

Properties

Caption: The text caption of the button.

Enabled: Whether or not the button is disabled.

Events

OnClick: Fired when the button is clicked

11.2 *CheckBox Control*

The CheckBox control allows you to use a regular checkbox for interaction with your scripts.

Properties

Caption: The text caption of the checkbox.

Enabled: Whether or not the checkbox is disabled.

Checked: Whether or not the checkbox is checked.

Events

OnCheck(variant checked): Fired when the checkbox is checked or unchecked



11.3 Edit Control

The Edit control allows you to support text-based user input without dependency on 3rd party controls.

Properties

Text: The text of the control

MultiLine: Whether the control is single line or multi, with scrollbars

ClientEdge: Whether the control has a client edge

Border: Whether the control has a border

BGColor: The background color of the box

TextColor: The color of text in the control

Events

OnKeyPress(key): Fired when a key is entered, returns the ASCII value of the key.

Methods

ScrollToEnd(): Scrolls to the bottom of the text.

11.4 ComboBox Control

The ComboBox control allows you to create a drop-down list of options from which the user can select.

Properties

BackColor: Sets or queries the color of the control background (as OLE_COLOR)

ForeColor: Sets or queries the color of the control text color (as OLE_COLOR)

SelectionBackColor: Sets or queries the background color of the selected item in the control (as OLE_COLOR)

SelectionForeColor: Sets or queries the text color of the selected item in the Control (as OLE_COLOR)

BorderVisible: Sets or queries whether or not the border of the control is visible (as Boolean)

Enabled: Sets or queries whether an object can respond to user-generated events (as Boolean)

Item(Item As Long): Sets or queries the text of a specific item in the control (as String)

ItemCount: Queries the number of items in the control (as Long)



ListIndex: Sets or queries the currently selected item in the control (as Long)

Text: Queries the text of the currently selected item in the control (as String)

ListVisible: Sets or queries whether the control's drop-down list is visible (as Boolean)

Events

OnSelect(Item As Long, string As String): Fired when an item in the list is selected

OnCloseUp(): Fired when the drop down menu is closed

OnDropDown(): Fired when the drop down menu is opened

Methods

AddItem (bstrItem As String): Inserts an item into the Control

ResetList(): Clears the Control

11.5 ListBox Control

The ListBox control allows you to display a box containing a list of options from which the user can select.

Properties

BackColor: Sets or queries the color of the control background (as OLE_COLOR)

ForeColor: Sets or queries the color of the control text color (as OLE_COLOR)

SelectionBackColor: Sets or queries the background color of the selected item in the control (as OLE_COLOR)

SelectionForeColor: Sets or queries the text color of the selected item in the Control (as OLE_COLOR)

BorderVisible: Sets or queries whether or not the border of the control is visible (as Boolean)

Enabled: Sets or queries whether an object can respond to user-generated events (as Boolean)

Item(Item As Long): Sets or queries the text of a specific item in the control (as String)

ItemCount: Queries the number of items in the control (as Long)

ItemHeight: Sets or queries the height of individual items in the ListBox (as Long)

ListIndex: Sets or queries the currently selected item in the control (as Long)

Text: Queries the text of the currently selected item in the control (as String)

Events



OnSelect(Item As Long, string As String): Fired when an item in the list is selected

Methods

AddItem (bstrItem As String): Inserts an item into the Control

ResetList(): Clears the Control



12 Scriptable E-Mail Notify plugin

DesktopX 3.0 supports scriptable plugins that simply add some custom functionality. In the case of the Scriptable Mail Notify plugin, only the actual email check functionality is offered by the plugin, while the account management, the preferences, the configuration and everything else is left to the DesktopX author to be implemented with DesktopX objects and scripts.

In order to use this plugin, you should add both a script and the “Scriptable Mail notify” plugin to an object.

The plugin exposes only one method and one notification event.

The method is:

Mail.CheckMail requestID, sLogin, sPass, sServer, accountType

requestID is a string identifying the request. Since this method is asynchronous and returns immediately, if you are handling multiple accounts you need to match responses with requests. This value let you do that, since you get back the same identifier when the response is ready.

sLogin is the mail account user name, sPass is the password, sSever is the server name (i.e. “mail.stardock.com”).

AccountType can be 0 for POP3 or 1 for IMAP.

Use the event wizard to get the following event added to the script:

```
Sub Mail_OnMailEvent(requestID, newMails)
```

This event will be fired when a request has been accomplished. RequestID contains the identifier passed in the request and newMails is a number indicating the unread mails.