# DOWNLOAD

Class CFlyout:

This class was originally designed to be similar to Window 8’s shell Flyouts; however, it has become something quite different.

CFlyout is intended to be a pretty dynamic class that can be window or control, much like the way ListView’s are used in this manner all over Windows OS (such as in right clicking on TaskBar icons). Internally, it consists of three main controls:

1. A ListBox (m\_vLB). The ListBox is used internally to help keep track of what text selection – it is hidden.
2. A Text control (m\_vFlyoutText). This is what is actually seen. The font may be customized completely. Wrapping of text and necessary adjustments to the GUI size are handled by the CFlyout.
3. A GroupBox (m\_vSelector). A highlighter for when CFlyout is not set to ReadOnly. It is not especially aesthetic as, whenever it is moved, as moving it requires a MoveDraw.

Methods:

public:

Show()

* Shows the flyout

Hide()

* Hides the flyout and sets the GroupBox selection to 1 (if needed).

OnMessage(msgs, sCallback="")

1. **Msgs** is a comma-delimited list of Window Messages. All messages are initially directed towards CFlyout\_OnMessage. Class-specific functionality, such as WM\_LBUTTONDOWN messages, are handled in that function.
2. **sCallback** is the function name of a callback for CFlyout\_OnMessage. sCallback must be a function that takes two parameters: a CFlyout object and a msg.

GetWidthAndHeight(ByRef riW, ByRef riH)

* Currently **iW** is simply set to m\_iW. Sometime in the future, **iW** will be automatically assigned based on a m\_iMaxWidth variable.
* Sets **iH** to what CalcHeight() returns.

CalcHeight()

* Calculates height of flyout based on the widest string in m\_asItems. Height will be no greater than **m\_iH**.

CalcHeightTo(iTo)

* Calculates height from **m\_iDrawnAtNdx** (topmost item being display) to item number **iTo**. Used in CFlyoutMenuHandler.

Move(bUp)

* Moves m\_vSelector up or down (based on **bUp**) by one item.

MoveTo(iTo)

* Moves m\_vSelector to item **iTo**.

Click(iMouseY)

* Moves m\_vSelector to the item rect within iMouseY. Does not work in ReadOnly mode.

GetCurSel()

* Returns currently selected item in flyout

GetCurSelNdx()

* Returns index of currently selected item in flyout

UpdateFlyout(aStringList = 0)

* Updates flyout with new items specified in **aStringList**; aStringList is assigned to m\_asItems. If aStringList is 0, then the flyout is redrawn using the same m\_asItems. If aStringList is non-zero, then potential changes in the flyout include changes in Height, and changes in X and Y positioning (depending on **m\_iAnchorAt**, **m\_bDrawBelowAnchor** and **m\_bFollowMouse**).

GUIEditSettings(hParent=0, sGUI="", bReloadOnExit=false)

* GUI interface for editing Flyout\_Config.ini. May be callable without initializing any CFlyout object like so, “**CFlyout.GUIEditSettings()**”
* **hParent** is for parentage GUI. If nonzero, then the parent window will be deactivated until the GUI is closed.
* **sGUI** is used to determine whether or not GUIEditSettings should be a standalone GUI with its own window or simply added to an existing GUI. i.e. (GUIEditSettings(hGUI1, “1”))
* **bReloadOnExit** : if true, Reload will be executed when the GUI is closed. This is useful if you are using multiple flyouts in one script and want them all to be updated with your latest changes.

private:

\_\_New() ; Note, hParent and asTextToDisplay excluded, any parameter that is set to 0 will instead be set via their corresponding key/value pair in Flyout\_config.ini. If a value still cannot be set, creation will likely fail.

1. hParent = 0. If nonzero, must be a valid handle to a window; when set, the CFlyout will become the child of hParent.
2. asTextToDisplay = 0. AHK [] linear array of strings. These will be displayed on the GUI. Each element of the array will be separated by a newline. If any element of the array is too wide, the text will be wrapped accordingly.
3. bReadOnly = 0. When true, the GUI is non-clickable, and no selection box (GroupBox) is provided. See Tooltip examples below.
4. bShowInTaskbar = 0. Typically used when CFlyout is used like a control instead of a window – you wouldn’t want your “control” showing up in the taskbar.
5. iX = 0. X coordinate. When iX **AND** iY are less than -32768, CFlyout will follow your mouse like a Tooltip; it wouldn’t make sense to make CFlyout non-readonly and also set it to follow your mouse, but you can anyway. See DictLookup for an example of how/why you would do this.
6. iY = 0. Y coordinate. Also needs to be set to be to less than -32768 in order for CFlyout to follow the mouse.
7. iW = 0. Width of the Flyout, in pixels. Text will be wrapped based on this number.
8. iMaxRows = 10. Determines the maximum height of the Flyout. Height is dynamically set based on the number of elements in asTextToDisplay. If iMaxRows is set to 10 and there are 11 elements in asTextToDisplay, then the 11th element will not show up on the Flyout; instead, it will can be scrolled down to and will be located beneath the 10th element, naturally.
9. iAnchorAt = -99999. Y Coordinates to “anchor” Flyout GUI to. When set to a number less than -32768, this is effectively telling CFlyout to not anchor to any point.
10. bDrawBelowAnchor = true. Completely ignored if iAnchorAt < -32768; when true, subsequent Flyout redraws/resizes will place the Top of the Flyout below the specified point; when false, it will place the Bottom of the Flyout above the specified point.
11. sBackground = 0. Background picture for Flyout. If 0 or an invalid file, then the background will be all Black.
12. sFont = 0. Font options in native AHK format sans color. For example, “Arial, s15 Bold”
13. sFontColor = 0. Font color in native AHK format (so it can be hex code or plain color like “Blue”)
14. sTextAlign = “Center”. Text alignment for CFlyout. Valid options are “Left”, “Right”, and “Center”.

\_\_Delete()

* Handles safe destruction of all objects CFlyout is responsible for. It is very important to note that, since CFlyout.FromHwnd stores *references* to CFlyout classes, any object that is assigned a Flyout (i.e. vFlyout := Object(CFlyout.FromHwnd[WinExist(“A”)])) must be released (i.e. vFlyout :=) in order for \_\_Delete to automatically be called upon removal of original Flyout object (assigned via vFlyout := new CFlyout(…))

\_\_Get(aName)

1. **aName** permutations are: GetFlyoutX, GetFlyoutY, GetFlyoutW, GetFlyoutH. These options are wrappers for WinGetPos of the flyout window.

LoadDefaultSettings(ByRef rsError)

* Loads settings for flyout from Flyout\_Config.ini. If any options are not specified, the default options specified in GetDefaultConfigIni will be used. If any unknown keys are in the inis, then an error is returned.
* **rsError** is set to an error message if the function returns false.

RedrawControls(bFollowMouseOverride=false)

* Called from **UpdateFlyout**. To update the flyout with new text, call **UpdateFlyout** instead.

GetCmdListForDisplay(iStartAt = 0)

1. Formats m\_asItems for display on m\_vFlyoutText control.

GetCmdListForListBox()

1. Formats m\_asItems for display on m\_LB control.

CalcAndSetSeperator()

* Currently unused. The idea is to fill a completely empty line a specified separator such as "-". Then flyout text, mainly in ReadOnly mode, could be made more readable being separated by “-”s. See DictLookup for an example of what I have in mind.

CalcBorderSelectionRect(s="")

* Calculates the selection rect for m\_vSelector needed to bound **s**. If **s** is not specified, then the current selection of the flyout is used.

MoveBorderSel(iInc)

* Moves m\_vSelector down **iInc** times.

EnsureCorrectDefaultGUI()

* Safety function to ensure that all GUI commands used by the class are directed towards the right GUI.

GetDefaultConfigIni()

* Default ini for Flyout\_Config.ini. Function is used is used for class\_EasyIni object to provide a safe way to push new sections and keys to Flyout\_Config.ini without changing any existing settings in Flyout\_Config.ini.

Public Member variables:

1. ; [Flyout] in Flyout\_config.ini
2. m\_iX : Stores iX from \_\_New.
3. m\_iY : Stores iY from \_\_New
4. m\_iW : Stores iW from \_\_New
5. m\_iMaxRows : Stores iMaxRows from \_\_New
6. m\_iAnchorAt : Stores iAnchorAt from \_\_New
7. m\_bDrawBelowAnchor : Stores bDrawBelowAnchor from \_\_New
8. m\_bReadOnly : Stores bReadOnly from \_\_New
9. m\_bShowInTaskbar : Stores bShowInTaskbar from \_\_New
10. m\_sBackground : Stores sBackground from \_\_New
11. m\_sTextAlign : Stores bTextAlign from \_\_New
12. m\_sFont : Stores sFont from \_\_New
13. m\_sFontColor : Stores sFontColor from \_\_New

Private Member variables:

1. m\_vConfigIni : class\_EasyIni object.
2. m\_bFollowMouse := false : Set to true when m\_iX and m\_iY are less than -32768
3. static m\_iMouseOffset := 16. Static pixel offset used to separate mouse pointer from Flyout when m\_bFollowMouse is true
4. m\_sSeperator := ; Not yet interfaced. The idea is to fill a completely empty line a specified seperator such as "-"
5. m\_vBorderSelRect : Stores dimensions of m\_vSelector
6. m\_iDrawnAtNdx : 0-based. Used to keep tracking scrolling position. If iMaxRows is set to 10, and 11 elements are in asTextToDisplay, and the user has scrolled to the 11th element, then m\_iDrawnAtNdx is set to 1, since we have scrolled past position 1.
7. m\_bIsHidden : True when Hide() is called. False when Show() is called.
8. m\_hFlyout : handle to CFlyout GUI
9. m\_hListBox : handle to invisible ListBox
10. m\_hFlyoutText : handle to Text control
11. m\_hGroupBox : handle to GroupBox selector
12. m\_hFont : handle to logical font for Text control
13. m\_hParent : handle to Parent assigned from hParent in \_\_New
14. m\_vFlyoutText : Control ID for Text
15. m\_vLB : Control ID for invisible ListBox
16. m\_vSelector : Control ID for GroupBox selector
17. m\_iFlyoutNum : Needed to support multiple CFlyouts.
18. m\_asitems : AHK Linear array formatted for Text control display purposes
19. m\_sCallbackFunc : Function name for optional OnMessage callbacks

Global helper functions:

CFlyout\_OnMessage(wParam, lParam, msg, hWnd)

* Wrapper for OnMessage. All window messages that need monitoring should be sent through this function instead of being sent directly to native OnMessage
* Message handler for all messages specified through **CFlyout.OnMessage**. Class-specific functionality, such as WM\_LBUTTONDOWN messages, are handled in this function.

CFlyout\_MouseProc(nCode, wParam, lParam, msg)

* Callback specified in **CFlyout.\_\_New** when **m\_bFollowMouse** is true.

GetRectForTooltip(ByRef riWndX, ByRef riWndY, iWndW, iWndH)

* Called only when **m\_bFollowMouse** is true. A reliable function to keep the flyout off of the mouse by 16 pixels, like how Tooltip works. It also keeps the flyout rect from spanning two or more monitors.
* **riWndX** is the desired X coordinate. It will always be incremented by 16 pixels. It will further be adjusted only if needed.
* **riWndY** is the desired Y coordinate. It will always be incremented by 16 pixels. It will further be adjusted only if needed.
* **iWndW** is the width of the flyout. WinGetPos is not called to retrieve the width instead because, when this function is called, the flyout may (in the future) be set to a different width than the current width.
* **iWndH** is the width of the flyout. WinGetPos is not called to retrieve the height instead because, when this function is called, the flyout may be set to a different height than the current height.

GetMonitorRectAt(x, y, default=1)

* Almost a copy of wp\_GetMonitorAt. It returns rect object for monitor which encompasses **x** and **y**. If the function fails, it returns the coordinates for **default**.

Anchor2(ctrl, a, d = false)

* My modification of a random anchor function that I found (using this one instead of Attach or Titan’s/Polythene’s Anchor v4 because this function, although the parameter syntax is downright atrocious, actually works in Windows 7 and 8).

LV\_SetDefault(sGUI, sLV)

* Sets the default GUI and ListView for use with GUI commands

LV\_GetSel()

* Wrapper to return the selected item number in a ListView

LV\_GetSelText(iCol=1)

* Wrapper to return the selected item number in a ListView

Going forward:

1. AnchorAt: Currently, the anchoring is pretty primitive. Since the Y coordinates need to be absolute, so it is easy to break. The absolute best way I can think of incorporating this functionality is to actually subclass the window to whatever thread it needs to be “anchored” to.
2. Convert to GDIP. I am not sure if this requires a *ton* of work or just a little. What I hope to accomplish by converting to GDIP is a way to avoid the flashing of the background picture everytime a redraw needs to happen. I also hope to incorporate a nice, rounded-rectangle GUI instead of the boring “–Caption” right-angle edge-ness of standard GUIs.
3. Convert CFlyout text /listbox controls to single, transparent ListView. I have already started work on this (see hyperlink). With this comes other great things such as, the removal of the GroupBox control that is used to highlight a selection – instead, a custom highlight color could be used; not only so, but it would also rectify the need to manually wrap text, like CFlyout does now.

How to use:

Examples: