MXSEditor Global Utility Plugin V1.0 – Document Version V1.1

Author: Josef Wienerroither
Modified: September 30th, 2018
Created: August 2011

VERSION HISTORY:
Version V1.0, DC20180930
- 3ds Max 2018 and 3ds Max 2019 builds added (Plugin version still 1.0)

Version V1.0, DC20160720
- Max 2008 to Max2017, x86 and x64 compatibility
- Callback functionality implementation
- Lots of other fixes and updates, Unicode compatibility
- Documentation added

Version V0.50 - Beta 1, DC20120214,
- First public release

AUTHOR:
Josef Wienerroither aka "spacefrog"
Web: www.frogsinspace.at
Email: spacefrog@chello.at

DOWNLOAD & INSTALL:
The most recent version can be downloaded here: 3sd Max MXSEditor Plugin - Current Release
Copy the appropiate file according to your 3ds Max version into your plugins folder and restart 3ds Max. The plugin should now be operational. To check this, type "**`showinterface MXSEditor`**" in the listener and hit enter. If a list of public properties and functions appears, the plugin got loaded.

FEATURES:
Maxscript Editor as Extended Viewport
Extensive control over the Maxscript Editor via maxscript
Access to the Scintilla/SciTE internal command set (see Scintilla Documentation)
Callback mechanism to get notified about changes and events occuring in the maxscript editor

USAGE:
Features for more general maxscript usage are documented below. For detailed documentation about the Scintilla/SciTE related functionality, please refer to the official Scintilla Documentation.

IMPORTANT:

The version of the Scintilla/SciTE codebase that is used for the maxscript editor is pretty outdated. Addionally it is heavily modified for the integration with 3ds Max. My investigation lead to the conclusion that it's based on codebase version 1.71, which was released back in August 2006. You may visit Scintilla's history page to read more about the progress made since then.

The following is important to remember when accessing Scintilla/SciTE related APIs using this plugin:
Many of today's Scintilla/SciTE features and documentation DO NOT apply to the maxscript editor. Only ever expect the featureset and API of V1.71. As todays online documentation targets the newer versions, it's recommended to use the docs bundled with the V1.71 distribution.

MXSEditor Interface

Main maxscript interface to control the editor,
"**`showinterface mxseditor`**" reveals all available methods and properties

```
  Interface: MXSEditor
   Properties:
    .x : integer : Read|Write
    .y : integer : Read|Write
    .width : integer : Read|Write
    .height : integer : Read|Write
    .enableCallbacks : boolean : Read|Write
    .isVisible : boolean : Read|Write
    .verbose : boolean : Read|Write
    .isExtendedVP : boolean : Read
    .mainHWND : HWND : Read
    .editHWND : HWND : Read
    .documentCount : integer : Read
    .currentDocument : string : Read
   Methods:
    <void>redraw()
    <void>show()
    <boolean>endExtendedVP()
    <void>reloadOptions()
    <boolean>editFile <string>filename quiet:<boolean>
       quiet default value: false
    <string>documentAtNumber <integer>index
    <void>menuCommand <string>menucode
    <boolean>setWindowPosition <int>x <int>y <int>width <int>height
    <string>getConfigProperty <string>name
    <integer>getConfigIntProperty <string>name
    <void>setConfigProperty <string>name <string>value
    <integer>listFunctions()
    <integer>listConstants()
    <integer>listProperties()
    <integer>listMenucodes()
```

Property	Description
`x, y, width, height`	Maxscript editor window position and size (if not in extended viewport)
`enableCallbacks`	quickly disable or enable installed callbacks without removing them
`isVisible`	Toggle maxscript editor visibilty
`verbose`	Toggle Scintilla commands verbose listener output
`isExtendedVP`	Maxscript editor is currently in extended viewport (readonly)
`mainWHND`	Maxscript editor‘s window handle
`editHWND`	Maxscript editor edit component‘s window handle
`documentCount`	Number of currently opened documents (tabs) in the maxscript editor
`currentDocument`	Full path and filename of the currently active tab‘s document

Methods:

`Redraw()`

Redraw the text content of the current visible tab

`Show()`

Opens the Maxscript editor window (see **`.isVisible`** property to hide)

`<boolean> EndExtendedVP()`

Detaches the maxscript editor from an extended viewport and restores the original editor window. Returns **`true`** if the editor was in an extended viewport, false otherwise.

`ReloadOptions()`

Reload Maxscript editor configuration files (eg. after a setting got changed externally)

`<boolean> EditFile <filename> quiet:<boolean>`

Open file in editor, if **`quiet:true`** is passed as optional argument, the usual modal dialog prompt is suppressed in case the file does‘nt exists

`<string> DocumentAtNumber <index>`

Returns the full path of the file loaded in editor‘s tab # <index> (1 based)

`<boolean> SetWindowPosition x y w h menuCommand`

Set maxscript editor‘s position and size using only one call

`MenuCommand <string>menuCode`

Sends the passed SciTE menucode the maxscript editor to perform the corresponding SciTE action.
For a quick list of available command codes, see **`ListMenucodes()`** below.
Note that the maxscript editor might not support all of those listed menucodes. For documentation of the SciTE menu codes, see the SciTE documentation here.

Examples:
`mxseditor.menucommand "IDM_FULLSCREEN"`   toggle editors fullscreen mode
`mxseditor.menucommand "IDM_TABSIZE"`   open the tab and indent size settings dialog

`<string> GetConfigProperty <name>`

Returns the string based property <name> from the current editor configuration. Those properties get read from the various editor configuration files at 3ds Max startup (eg. MXS_Editor.properties). If the property does‘not exist, and empty string ("") is returned.

Example:
`GetConfigProperty "font.base“` returns the configured base font as string value

`<int> GetConfigIntProperty <name>`

Returns the property <name> from the current editor configuration as integer. If the property is not an integer a value of 0 is returned. If the property does'not exist -1 is returned.

Example:
`GetConfigIntProperty "caret.width"` returns the cursor's width in pixels

`SetConfigProperty <string> <string>`

Sets the property <name> from the current editor configuration to the passed string value. Please note that this is of limited use, as the new property value does not become active before 3ds Max is restarted. As the value does'nt get saved to the configuration file, it will be lost after a restart.

`ListFunctions()`

Lists all SciTE internal commands which can be used to control the maxscript editor component. See the documentation or the **`MXSEFuncs`** Interface below. For a description of the individual SciTE keycommands, see the official online documentation here and here.
Note that not all commands might be implemented or supported in the maxscript editor.

`ListConstants()`

Lists all SciTE internal constants and there corresponding integer value to be passed to the **`MXSFuncs`** and **`MXSProps`** interfaces. See the documentation for **`MXSEFuncs`** and **`MXSProps`** below. For a description of these constants see official Scintilla documentation here.
Note that not all constants might apply or be supported in the maxscript editor.

`ListProperties()`

Lists all SciTE internal properties and their signature available to query and change the current configuration and settings of the maxscript editor using the **`MXSProps`** interface. See the documentation for **`MXSProps`** below. For a description of those properties, see the official online documentation here. Note that not all properties might apply or be supported in the maxscript editor.

`ListMenucodes()`

Lists all available SciTE menu commands available which can be sent to the maxscript editor. See the documentation for the **`MenuCommand()`** method above. Note that the maxscript editor might not support all of those listed menu commands. For documentation of the SciTE menu commands, see the SciTE documentation here.

MXSEFuncs struct

Maxscript struct to access Scintilla's editor functions. Entering "**`MXSEFuncs`**" in the listener reveals two public methods

```
#Struct:MXSEFuncs(
  call:<fn>; Public,
  list:<fn>; Public)
```

Methods:

`List()`

Lists available SciTE internal editor commands and their parameter signature. For a description of the individual SciTE keycommands, see the official online documentation here and here.

Note that not all commands might be implemented or supported in the maxscript editor.

`Call()`

Calls a SciTE internal editor keycommand. The first argument is the SciTE command name passed as string value. Addional arguments depend on the specific SciTE's function signature. See **`MXSFuncs.List()`** below for the various command's arguments and their data types. Command parameters and return values are past as their respective maxscript equivalents.

Example:
To set the editor selected text's background color we use **`MXSFuncs.List()`**or the SciTE online documentation to find the command **`void SetSelBack(bool, color).`**

Thus we use **`MXSFuncs.call()`** to actually execute the command in the listener:

```
mxsefuncs.call "SetSelBack" true new (color 100 200 30 128)
```

(see http://www.scintilla.org/PaneAPI.html for a description of the boolean parameter...)

Maxscript variables can be passed by reference with commands that expect a variable to be filled with additional return data (eg. the selected text)

Example:
To receive the currently selected text we use the following mxsefuncs call

```
mxsefuncs.call "GetSelText" &myText
```

The Maxscript variable **`myText`** now will contain the currently selected text

MXSEProps struct

Maxscript struct to access Scintilla's editor properties. Entering "**`mxseprops`**" in the listener reveals thw following public methods

```
#Struct:MXSEProps(
  set:<fn>; Public,
  list:<fn>; Public,
  get:<fn>; Public)
```

Methods:

`List()`

Similar to **`mxsefuncs.list()`**, this one lists available SciTE config properties. They essentially represent the current internal settings and state of the editor component. t's important to recognize that those are NOT properties defined in the external scintilla "options" files .
See the official online documentation here.

Note that not all properties might be implemented or supported in the maxscript editor.

`Get <propertynamestring> [<optional argument>]`

Returns the current value of the corresponding property. See **`mxseprops.List()`** for available properties, their return datatypes and the optional argument

`Set <propertynamestring> <value>`

Sets the current property value. See **`mxseprops.List()`** for available propertynames and their value type (the leftmost column of the output).

Examples:

To change the current zoom level of the editor text (default value is 0)

```
mxseprops.set "zoom" 4
```

Or to select the current editor tab's text from character position 10 to position 150

```
mxseprops.set "selectionstart" 10
mxseprops.set "selectionend" 150
```

MXSECallback struct

Maxscript struct to access the callback framework. Callbacks hook into maxscript editor notifications. This includes events like text and editor input focus changes, autocomplete selections, mouse input events etc...

```
#Struct:MXSECallback(
  set:<fn>; Public,
  show:<fn>; Public,
  Enable:<fn>; Public,
  remove:<fn>; Public,
  list:<fn>; Public,
  removeAll:<fn>; Public,
  getCallbackText:<fn>; Public,
  get:<fn>; Public,
  getCallbackData:<fn>; Public,
  Disable:<fn>; Public)
```

Methods:

`Enable(), Disable()`

Enable/disable the callback notification functionality globally. Does not modify the list of registered callbacks, but provides a way to quickly enable and disable the whole callback mechanism.

`Set <event id> <maxscript function>`

Registers the corresponding maxscript function to be called when respective notification is triggered. For a list of valid event ids, please see the **List()** method's documentation below.

`Get <event id>`

Get the maxscript function that is registered to be called by the the corresponding notification event.

`List()`

List all available and supported SciTE notification event ids.

```
#StyleNeeded                          #URIDropped
#CharAdded                            #DwellStart
#SavePointReached                     #DwellEnd
#SavePointLeft                        #Zoom
#ModifyAttemptReadOnly                #HotSpotClick
#DoubleClick                          #HotSpotDoubleClick
#UpdateUI                             #CallTipClick
#Modified                             #AutoCompleteSelection
#MacroRecord                          #KillFocus
#MarginClick                          #SetFocus
#NeedShown                            #Change
#Painted
#UserListSelection
```

`Remove <event id>`

Removes the given event id from the list of registered notification events. The corresponding maxscript   function will no longer be called when the notification event occurs.

`RemoveAll()`

Removes all maxscript functions from the notification callbacks. Though the notification callback mechanism is essentially disabled, notification events are still processed internally. Should be followed by **`Disable()`** (see above), to completely disable the notification callback mechanim.

`Show()`

List current installed SciTE notification callback Ids and their respective maxscript function.

`GetCallbackData()`

To be called inside a registered notification callback function. Returns an array holding 19 data elements which form a Scintilla SCNotification struct. For further documentation about this struct's content and which fields hold valid data for a given callback, read the official documentation here. This includes exact documentation about which fields get used with which notification.

The elements in the array, their corresponding **`SCNotification`** struct counterparts and the event ids which use those fields to pass data. This might be not complete, so it's always good to experiment and check for other eventId/valid field combos. It might be feasable to analyze the relevant SciTE source code (SciteBase.cxx etc.) to learn about addional callback data passing.

1: **`idFrom: Integer,`** ControlID which sent the notification

2: **code: Integer**, the raw SCN_* notification code (see **`SCNotification`** struct documentation)

3: **position: Integer,** character position value
used with: **`#StyleNeeded, #DoubleClick, #Modified, #DwellStart, #DwellEnd, #CallTipClick, #HotSpotClick, #HotSpotDoubleClic,`**

4: **ch: Integer**, character entered, used with: **`#CharAdded`**

5: **modifiers: Integer,** modifiers flags, used with: **`#HotSpotClick, #HotSpotDoubleClick`**

6: **`modificationType: Integer,`** type of modification, used with: **`#Modified`**

7: **text: Integer64,** memory ADDRESS of textbuffer or **`undefined`**
for performance reasons, this field only holds the ADDRESS if there is actual text available with the notification or **`undefined`** otherwise. To retrieve the text, see **`GetCallbackText()`** below.
Used with: **`#Modified, #UserListSelection, #AutoCompleteSelection`**

8: **length: Integer,** length value used by certain notifications, used with: **`#Modified`**

9: **linesAdded: Integer,** number of lines added , used with: **`#Modified`**

10: **message: `Integer,`** message code, used with: **`#MacroRecord`**

11: **wParam: Integer,** wParam for **message**, used with: **`#MacroRecord`**

12: **lParam: Integer,** lParam for **message**, used with: **`#MacroRecord`**

13: **line: `Integer,`** used with: **`#Modified, #DoubleClick`**

14: **`foldLevelNow: Integer,`** current foldlevel applied to the line, used with: **`#Modified`**

15: **`foldLevelPrev: Integer,`** previous foldlevel for the line, used with: **`#Modified`**

16: **margin**: **`Integer,`** id of the margin that was clicked on, used with: **`#MarginClick`**

17: **listType**: **`Integer,`** `list`type picked , used w.: **#UserListSelection, #AutoCompleteSelection**

18: **x: Integer**, horizontal pixel position , used with: **`#DwellStart, #DwellEnd`**

19: y**: Integer**, vertical pixel position, used with: **`#DwellStart, #DwellEnd`**

`GetCallbackText()`

To be called inside a registered notification callback function. Retrieves and returns text passed by the current notification, if the notification supports that. Returns **`undefined`** in the other case.