Construction Set Extender

shadeMe

Version 5.0

Table of Contents

Changes to existing tools

- **Creation and modification of master files:**
  Master files can be actively edited and saved in the CS by setting them as active plugins. They will retain their master file status upon saving.
- **Modification of master file header info:**
  The Author and Description fields of master files are no longer disabled by default and can be edited like any other plugin file.
- **Removal of the need for mod de-isolation:**
  The CS will now automatically save loaded ESP files as masters of the active plugin.
- **Enhanced Find Text tool:**
  Entries in the find text dialog can directly be invoked for editing, i.e., double clicking the results of a search will bring up the corresponding item's dialog box or load the object into the render window if it is a reference.
- **Loading of plugins with missing masters:**
  The CS no longer exits with an error message when a loaded plugin has a missing master. It will now skip the master altogether.
- **Door markers have property dialogs:**
  Double clicking on door markers will now bring up their reference properties dialog box.
- **Warning message boxes are forever banished:**
  Warnings generated by the CS are now logged to the new Console window.
- **Fast Exit for the CS:**
  Similar to the FastExit plugin for the runtime; quickly exits the CS.
- **Saving plugins as ESM files:**
  The CS can now save plugins as either ESP or ESM files.
- **Enhanced Save Script confirmation box:**
  The confirmation message box now has a cancel option.
- **Enhanced Recompile All Scripts tool:**
  The recompile all scripts tool now only parses scripts in the active plugin.
- **Logging of recompile results:**
  The results of a recompile operation are logged to the Console.
- **Icons with mipmaps are allowed as usable textures:**
  Icons with mipmaps can be previewed correctly and the CS no longer generates errors about the matter.
- **Unknown record and group types are allowed:**
  Plugins containing unknown records, sub records or group types no longer crash the CS.
- **Maximum compiled script size doubled:**
  Compiled bytecode size has been increased to 32KB.
- **Script compiler errors accumulate:**
  Compiler errors are displayed on a per-script basis.
- **Saving plugins when there are open dialogs:**
  The "Allow Unsafe Saves" CS INI setting is permanently enabled.
- **BSA Archives are no longer loaded selectively:**
  All BSA archives in the Data folder are loaded at startup, regardless of their connection to an active plugin.

Fixes for existing bugs

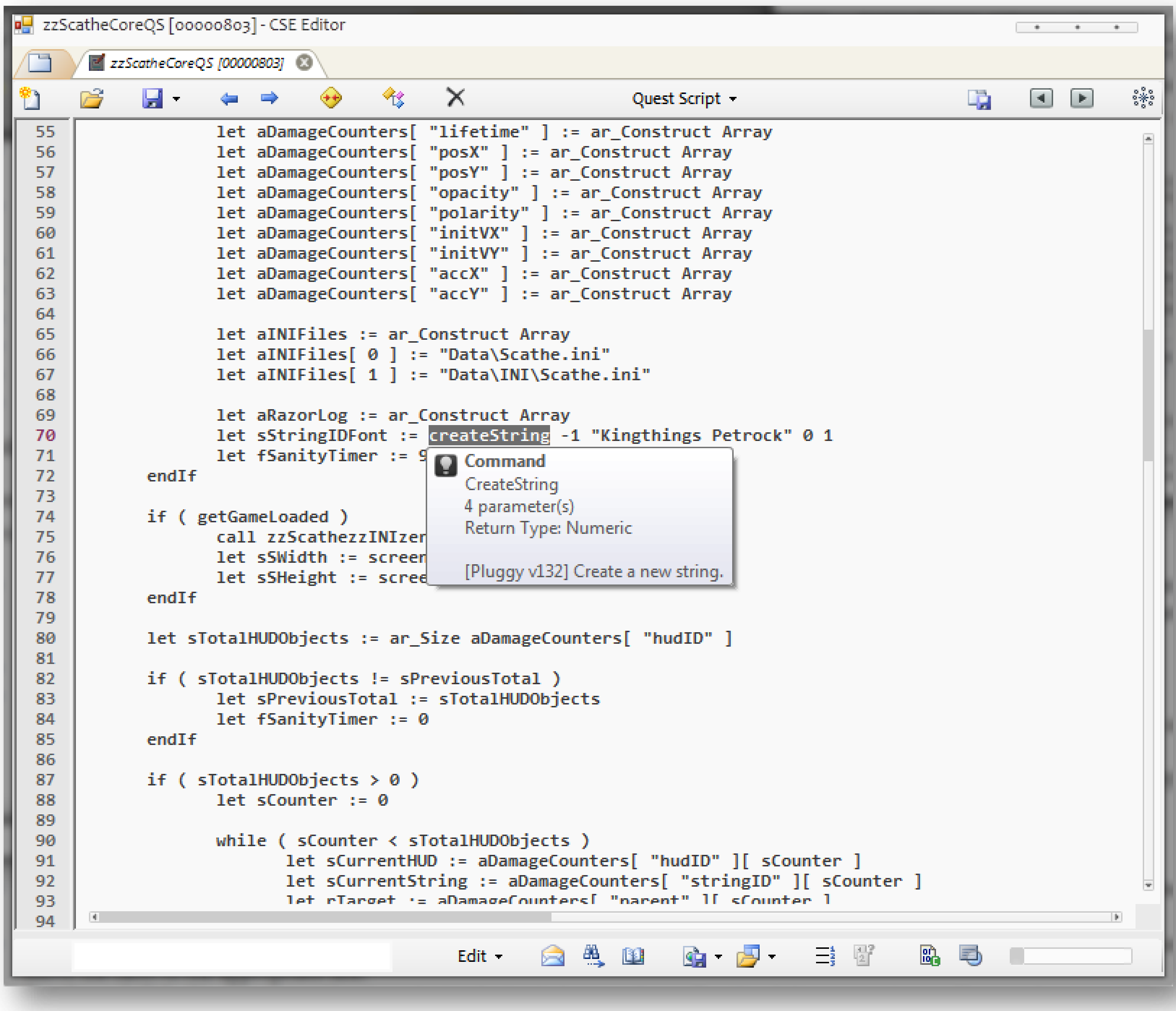
- **Response editor microphone bug:**
  Fix for the CTD that occurs on machines with Realtek soundcards, when the response window is initialized without a microphone plugged in its socket.
- **Topic info data reset bug:**
  Fix for the bug that automatically clears result script text and info flags if they are visible when a new topic is added to the topic list.
- **Face-Gen bug:**
  Fix for the CTD that occurs due to the improper initialization of the facegen renderer in NPC and creature dialogs.
- **Identical-to-master dialog and worldspace edit bug:**
  Fix for the version control related bug that makes unnecessary edits to cells, dialogs and worldspaces should one of the plugin's masters have an active record of the same.
- **Race description dirty-edit bug:**
  Fix for the bug that copies race description from one race record to another if the latter were to invoke the spell checker.
- **Pink water bug:**
  Not a bug per-se, but what the heck! Fix for the ugly pink water used by default.
- **BS-Assertion bug:**
  This bug is deep rooted in the editor code and tends to cause a fairly large number of CTDs for no reason.
- **Render window menu item bug:**
  Fix for the bug that prevented the Render window for being closed when using the View > Render Window main menu item.
- **Topic info copy bug:**
  Fix for the bug that caused the wrong topic info record to be flagged as modified during a copy operation.
- **Lip Sync Generator:**
  The infamous lip sync tool has been finally fixed! More details in the next section.

New tools

- **Script Editor:**

  The CSE Editor is a complete replacement for the CS' vanilla script editor. It has been written from scratch and is basically superior to the vanilla in every way. Its design is supposed to be intuitive enough to allow even new CS users to get used to it and its many advanced features.

  I'd denote important points of interest in the screenshot below but my Photoshop skills are all but extant.


zzScatheCoreQS [00000803] - CSE Editor
zzScatheCoreQS [00000803]
Quest Script
Edit

```
        let aDamageCounters[ "lifetime" ] := ar_Construct Array
        let aDamageCounters[ "posX" ] := ar_Construct Array
        let aDamageCounters[ "posY" ] := ar_Construct Array
        let aDamageCounters[ "opacity" ] := ar_Construct Array
        let aDamageCounters[ "polarity" ] := ar_Construct Array
        let aDamageCounters[ "initVX" ] := ar_Construct Array
        let aDamageCounters[ "initVY" ] := ar_Construct Array
        let aDamageCounters[ "accX" ] := ar_Construct Array
        let aDamageCounters[ "accY" ] := ar_Construct Array

        let aINIFiles := ar_Construct Array
        let aINIFiles[ 0 ] := "Data\Scathe.ini"
        let aINIFiles[ 1 ] := "Data\INI\Scathe.ini"

        let aRazorLog := ar_Construct Array
        let sStringIDFont := createString -1 "Kingthings Petrock" 0 1
        let fSanityTimer := 9
    endIf

    if ( getGameLoaded )
        call zzScathezzINIzer
        let sSWidth := screen
        let sSHeight := scree
    endIf

    let sTotalHUDObjects := ar_Size aDamageCounters[ "hudID" ]

    if ( sTotalHUDObjects != sPreviousTotal )
        let sPreviousTotal := sTotalHUDObjects
        let fSanityTimer := 0
    endIf

    if ( sTotalHUDObjects > 0 )
        let sCounter := 0

        while ( sCounter < sTotalHUDObjects )
            let sCurrentHUD := aDamageCounters[ "hudID" ][ sCounter ]
            let sCurrentString := aDamageCounters[ "stringID" ][ sCounter ]
            let rTarget := aDamageCounters[ "parent" ][ sCounter ]
```

```
Command
CreateString
4 parameter(s)
Return Type: Numeric

[Pluggy v132] Create a new string.
```

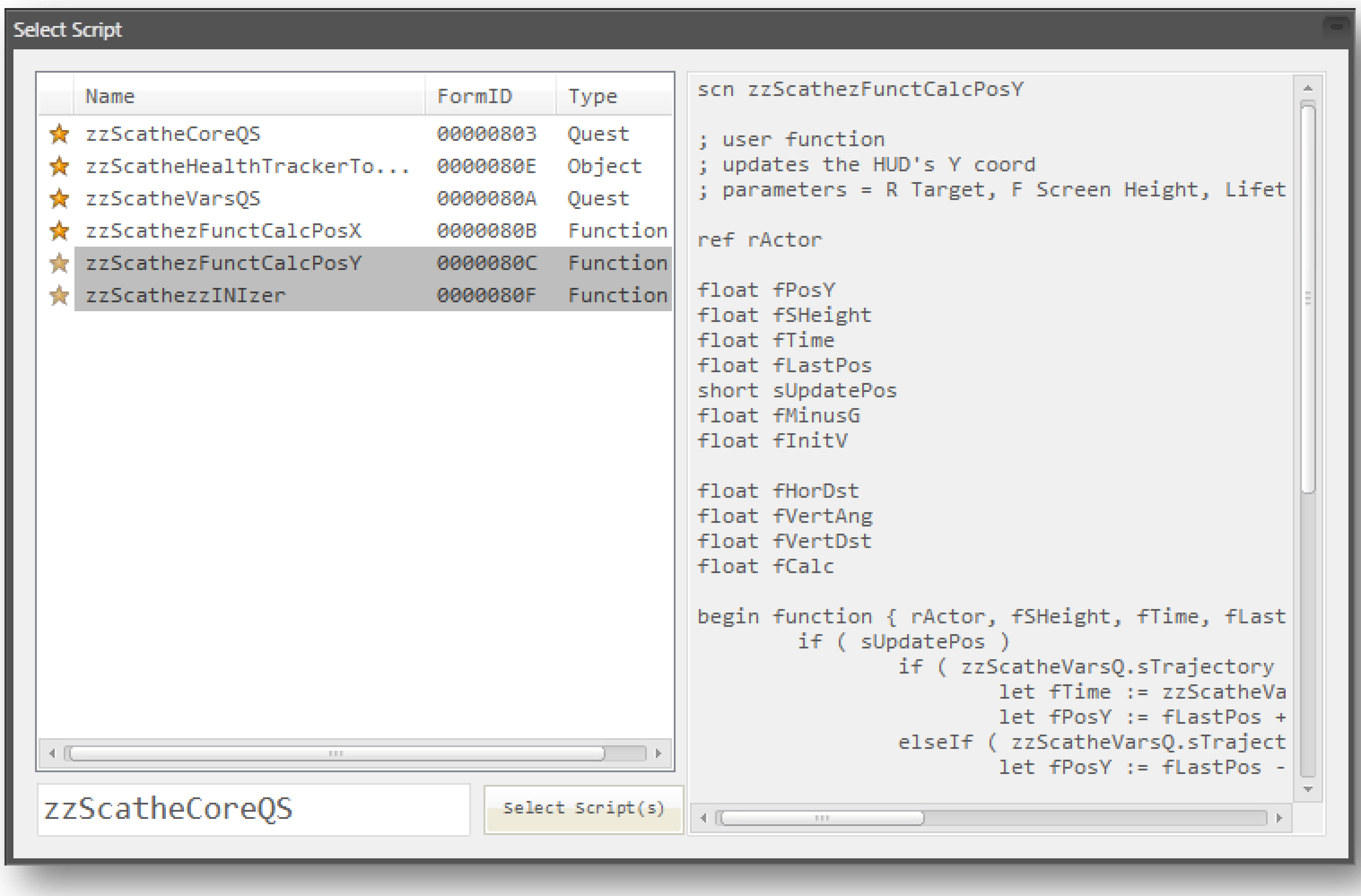
The CSE Editor is a tabbed script editor, as obvious as it may seem. Each editor window is called a tab container while the actual editor component (the one with the text editor control) is called a workspace. A tab container is always spawned with an empty workspace (think of a workspace as a vanilla editor). It can hold an arbitrary number of workspaces and allows operations such as tab rearranging and tab tearing. The very first tab seen above is the 'New tab' button; it creates a new workspace in its tab container.

Tabs can be rearranged by dragging them across the tab strip. Tab tearing too is centered on the tab strip. There are two operations related to tab tearing:

1. **Allocation** – This involves dragging a tab out of its tabstrip and elsewhere. This spawns a new tab container at the point of release and moves the torn tab into it.
2. **Relocation** – Tabs can be moved b'ween tab containers seamlessly. This is done by tearing a tab and releasing it at another tab strip. If done correctly, the torn tab will be relocated to its new container.

On to the workspace - This is the primary component of the editor. Every component below the main tool bar is a child of the workspace. Each workspace is isolated, meaning each has its own controls (text editor, line number margin and such). Here are the list of the buttons/controls and their significance in the order of appearance (New button – Progress bar in the bottom right):

- ***New*** – Creates a new script. On initial use in a workspace, all controls become enabled and usable.
- ***Open*** – Opens the 'Select Script' dialog for script selection. More on this:



Name	FormID	Type
zzScatheCoreQS	00000803	Quest
zzScatheHealthTrackerTo...	0000080E	Object
zzScatheVarsQS	0000080A	Quest
zzScathezFunctCalcPosX	0000080B	Function
zzScathezFunctCalcPosY	0000080C	Function
zzScathezzINIzer	0000080F	Function

zzScatheCoreQS

```
scn zzScathezFunctCalcPosY

; user function
; updates the HUD's Y coord
; parameters = R Target, F Screen Height, Lifet

ref rActor

float fPosY
float fSHeight
float fTime
float fLastPos
short sUpdatePos
float fMinusG
float fInitV

float fHorDst
float fVertAng
float fVertDst
float fCalc

begin function { rActor, fSHeight, fTime, fLast
        if ( sUpdatePos )
                if ( zzScatheVarsQ.sTrajectory
                        let fTime := zzScatheVa
                        let fPosY := fLastPos +
                elseIf ( zzScatheVarsQ.sTraject
                        let fPosY := fLastPos -
```

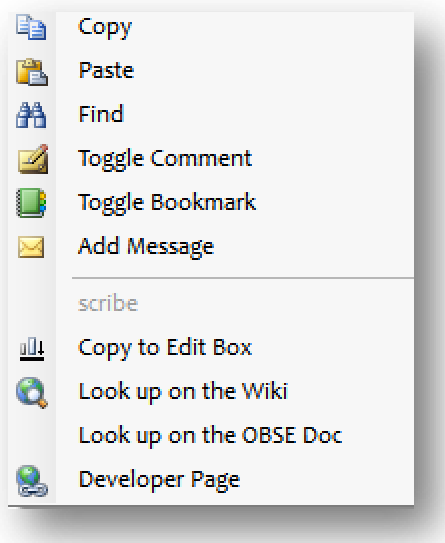
The list view displays all the loaded scripts and the text box to the right shows a preview of the selected script. The textbox at the bottom can be used to select a particular script by its editorID or formID. Multiple scripts can be selected in the list view during an 'Open' operation and this is where the 'Select Script(s)' button comes to play. The selected scripts are loaded neatly into their own tab on clicking it. The list view can be sorted by each column. The first column denotes the state of each script – A golden star denotes that the script is from an active plugin and an 'X' mark denotes that the script is deleted. The list view is sorted by status on initialization.

- ***Save*** – Attempts to compile and save the loaded script. This button has a drop down
  - ***Save script but don't compile*** – This does as it says. On loading a non-compiled script, the editor will generate an error warning you about the script's status.
  - ***Save script and active plugin*** – This tool is mostly a luxury. It attempts to compile and save the loaded script but saves the active plugin regardless of the compilation result.
- ***Previous*** – Loads the previous script, if any.
- ***Next*** – Loads the next script, if any.
- ***Recompile all active scripts*** – Attempts to compile and save every script in the active plugin. Failed compilation results in an entry in the console.
- ***Recompile dependencies*** – Attempts to compile and save any scripts (regular and result scripts) that might reference the loaded script and prints a detailed report to the console.
- ***Delete*** – Opens the 'Select Script' dialog for the selection of the script to be deleted.
- ***Script type*** – Selects the loaded script's type.
- ***Save all open scripts*** – Attempts to compile and save every script in the tab container.
- ***Navigate back*** – Jump back in the navigation stack (more on it later).
- ***Navigate forward*** – Jumps forward in the navigation stack.
- ***Preferences*** – Opens the preferences window. Some changes may require a restart of the editor.

The 'Modified' status of a script is shown in its tab – A dark icon represents 'No change', a colored icon denotes 'Script changed'.

The bottom status bar is actually a splitter bar which can be moved to resize the editor area and show the controls beneath it. The buttons on it are as follows:

- ***Common textbox*** – Take input for the various functions in the Edit menu.
- ***Edit menu***:
  - ***Find*** – Find all instances of a string in the loaded script. Results are displayed in a list view below the bottom status bar and pointers are placed at the locations of the query text.
  - ***Replace*** – Replaces all instances of a string in the loaded script. The search string must first be entered, and then the replace string. The replacement string can be a null string, in which case all occurrences of the search string will be purged.
  - ***Go to Line*** – Jumps to the given line number. This tool cannot be used in the offset viewer.
  - ***Go to Offset*** – Jumps to the given script offset. This tool can only be used in the offset viewer.
- ***Message list*** – Standard output for the script validator, preprocessor and the compiler. Custom messages are also displayed here. Double clicking on a non-message item will move the caret to the appropriate line; otherwise remove it from the list.
- ***Find results*** – Displays a find operator's results. Closing this list view will also remove all placed pointers. Double clicking on an item will move the caret to the appropriate line.
- ***Bookmark list*** – Displays stored bookmarks for the loaded script, if any. Double clicking on an item will move the caret to the appropriate line.
- ***Dump script*** – Saves the loaded script as a file of arbitrary type in a selected folder.
  - ***Dump all tabs*** – Saves all open scripts to a selected folder as text files.
- ***Load script*** – Loads a plain text type file from disk into the editor. Replaces any existing content.
  - ***Load multiple scripts into tabs*** – Loads multiple scripts into a tab of their own.
- ***Fetch variable indices*** – Attempts to compile and save the loaded script. If successful, enumerates every variable in the script and its type and index. Indices can be edited by double clicking on the desired index cell.
- ***Update variable indices*** – Only used in conjunction with the above tool. Updates the script with the modified (if any) variable indices. This is an advanced tool and must be used with care as it can easily break scripts.
- ***Toggle Offset Viewer*** – Displays line offsets in place of line numbers. These offsets are useful when debugging OBSE errors as they only mention script offsets. This tool may only be used with compiled scripts.
- ***Toggle Preprocessed Text Viewer*** – Preprocesses the script text and displays it in a separate text viewer.
- ***Progress bar*** – Show the compiled byte code size of the loaded script.



The editor's context menu offers quick access to some of its features. 'Toggle comment' places or removes a semicolon at the line under the mouse cursor. 'Toggle bookmark' allows the creation of bookmarks based on particular lines. Since they are saved with each script, every script can have its own set bookmarks. 'Copy to edit box' copies the text under the mouse cursor to the common text box. Find quickly searches for the same. 'Developer Page' is a special menu item that appears on select keywords. In most cases, such keywords turn out to be command/function identifiers. OBSE plugin writers can inter-operate with CSE to add specific links to each of their commands though this interface. Another special menu item is the 'Jump to …' button. It appears when the text under the mouse cursor is a legal scriptable object or script (like a quest or an object reference). Clicking it can either create a new workspace and load the jump script if it isn't open in the parent tab container, or switch to the workspace that has the script loaded. Each jump is tracked internally by a navigation stack. The main tool bar's 'Navigate Forward' and 'Navigate Backward' can be used in this context. The 'Add Message' tool can be used to add notification messages that will show on script load. Existing messages can be deleted by double clicking them in the message list.

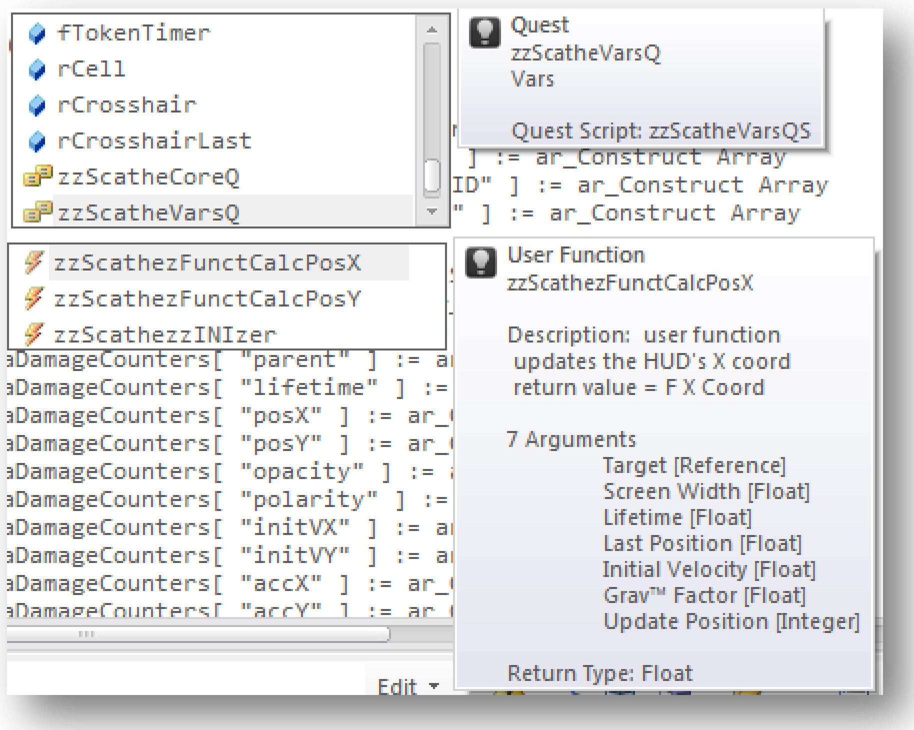
The CSE editor has further more features:

- ***Auto-indentation*** – Script lines are automatically indented and outdented depending upon the script structure. The script writer can carry on with coding without having to worry about indentation.
- ***Toggle comment*** – Single and multiple line(s) can be commented in/out with this tool. It uses the first line in its selection as a reference to the operation it performs i.e., if the first line in the selection is already commented, the operation will uncomment all of the selected lines.
- ***Line limit indicator*** – The editor displays an indicator when the number of characters in a script line exceeds the maximum limit of 512.
- ***Tab indentation*** – Lines can be batch indented and outdented by using the *Tab* key with and without the *Shift* key modifier.
- ***Persistent Fonts*** – Not much of a feature but something that many have pined for.

Beyond the above, the editor has 3 main components to it:

- ***IntelliSense***
- ***Script Validator***
- ***A preprocessing engine called 'A preprocessing engine'***

INTELLISENSE



```
fTokenTimer
rCell
rCrosshair
rCrosshairLast
zzScatheCoreQ
zzScatheVarsQ
```

Quest
zzScatheVarsQ
Vars

Quest Script: zzScatheVarsQS

```
] := ar_Construct Array
ID" ] := ar_Construct Array
" ] := ar_Construct Array
```

```
zzScathezFunctCalcPosX
zzScathezFunctCalcPosY
zzScathezzINIzer
aDamageCounters[ "parent" ] := a
aDamageCounters[ "lifetime" ] :=
aDamageCounters[ "posX" ] := ar_
aDamageCounters[ "posY" ] := ar_
aDamageCounters[ "opacity" ] :=
aDamageCounters[ "polarity" ] :=
aDamageCounters[ "initVX" ] := a
aDamageCounters[ "initVY" ] := a
aDamageCounters[ "accX" ] := ar_
aDamageCounters[ "accY" ] := ar_
```

User Function
zzScathezFunctCalcPosX

Description: user function
updates the HUD's X coord
return value = F X Coord

7 Arguments
Target [Reference]
Screen Width [Float]
Lifetime [Float]
Last Position [Float]
Initial Velocity [Float]
Grav™ Factor [Float]
Update Position [Integer]

Return Type: Float

This component provides easy referencing of commands, local variables, remote scripts and their variables. The component maintains a database of usable commands/functions, vanilla or OBSE-related, user defined functions and quests. This database is updated every 10 minutes to keep recent changes accessible to the user. The interface it uses is similar to Microsoft Visual Studio's – A context specific list of reference items. The interface popup will appear after a custom number of characters following a delimiter (adjustable in the Preferences window) and will filter its items as you type. When visible, the *Up* and *Down* keys are used to navigate the list and the *Tab* key to insert the selected item. The *Escape* key can be used to close the popup. Each item has a description of its down that is determined by its type:

- **Commands** – Descriptions include command alias, description, number of parameters, command source and return type. Those that require a calling reference aren't displayed unless the last typed identifier is a dot operator and the calling reference is an object placed in the world or a reference variable.
- **Variables** – Descriptions include delimited comments following their declaration and their type. E.g. `short sSomeVar     ; Stores some value`
- **Quests** – Descriptions include the name field of the quest and the editorID of the quest script, if any.
- **User Defined Functions** – Take the following UDF script

```
Scn test

; this is an UDF script
; some text - foo
; more foo

Short sArg1  ; Some arg
Float fArg2  ; Another arg

Begin function {sArg1 fArg2}
      Let sArg1 := 111
      Setfunctionvalue sArg1
End
```

The description will include the comment text b'ween the script name declaration and the first local variable's. Arguments are treated as variables and enumerated. And finally, the return type of the UDF is stated.

Furthermore, IntelliSense tracks what you type and displays the interface list in a fairly intelligent manner. It supports 3 identifiers:

- **Set** or **Let** – Both of these populate the list with local variables and quests.
- **Call** – Populates the list with user defined functions.
- **Dot** (.) – Context sensitive. If the identifier before the dot operator is a scriptable object, such as a quest, it will display its script's variables and also commands that require a calling reference (if the calling reference is an object placed in the world or a reference variable).

IntelliSense allows the quick lookup of a valid identifier by double clicking it; this brings up an info tip describing it.

SCRIPT VALIDATOR

The script validator catches errors that the vanilla script compiler doesn't. The following errors are those that are caught:

- Invalid block types for non-object scripts.
- Script name re-declarations.
- Superfluous expressions in commands.
- Identifiers that start with integers.
- Nested variable declarations.
- Variable re-declarations.
- Unreferenced variables.

The token parser expects operators, operands and function arguments to be delimited by one of the following characters: `., (){}[]\t`

A PREPROCESSOR ENGINE

The editor has a preprocessor engine that allows script writers to use various preprocessor directives similar to Visual Studio's. All directive declarations/definitions need to be represented as comments. Preprocessor directives are grouped in two: Single and multi-line directives. Single line directives do not exceed a line of code in the text editor. Such directives use the '#' character as their prefix. Multi-line directives, on the other hand, encompass multiple lines of code and must be prefixed with the '@' character. The multi-line argument/value needs to be enclosed in curly braces. Some directives may support a single encoding type.

For example:

```
;#DEFINE MACRO_FOO "FOO~POO"

;@IF (MACRO_FOO != 123.222 || (MACRO_FOO < 10 && MACRO_FOO > 4.2))
;{
;     PRINT "MACRO CONDITION EVALUATED AS TRUE!"
;}
```

- **Define** – Defines a preprocessor macro, similar to VS'. Macro identifiers can only contain alpha-numerics and underscores and are case sensitive. They must be delimited with one of the following chars to be recognized: `., (){}[]\t.` Macro values can have any character and aren't limited by delimiters. They can be used in any context as the preprocessor simply replaces the macro identifier with its value before compilation. For instance,

```
;#DEFINE _DEBUG 1

IF _DEBUG
   PRINT "THIS MESSAGE WILL BE PRINTED IF _DEBUG IS SET TO A NON-ZERO VALUE"
ENDIF

;@DEFINE PRINTMESSAGESTRING
;{
; PRINT "MESSAGEONE!"
; PRINT "MESSAGETWO!"
; ; COMMENT
;}

IF ZZQUEST.VAR == 1
      PRINTMESSAGESTRING
ENDIF
```

  Standard macros can be defined in the 'STDPreprocDefs.txt' file. These macros are always parsed during each preprocessor operation.

- **Import** – Allows external text to be inserted into scripts, similar to #include in VS. The text files to be inserted must be present in the 'Data\Scripts' directory. Consider the following example,

```
DATA\SCRIPT\TESTSNIP.TXT
      FLOAT  FQUESTDELAYTIME
      SHORT  DOONCE
      LONG   GOLDVALUE

SCRIPT ZZTESTQS
      SCN ZZTESTQS

      ;#IMPORT "TESTSNIP"

      BEGIN GAMEMODE
            PRINT "FOO"
      END

PREPROCESSED SCRIPT
      SCN ZZTESTQS

      FLOAT  FQUESTDELAYTIME
      SHORT  DOONCE
      LONG   GOLDVALUE

      BEGIN GAMEMODE
            PRINT "FOO"
      END
```

  The Import directive is recursive, so imported scripts/snippets can have their own preprocessor directives. It does not support multi-line encoding.

- **Enum** – Defines an enumeration (enum for short). An enumeration is basically a single line definition that allows multiple macros to be defined in order. Enum items can only have numeric values. They need not be continuous i.e., an item may be declared without an initialization value, in which case it will be assigned one more than the value of its predecessor. The default value starts with 0. The syntax for an enumeration is as follows:

```
;#ENUM ENUM_NAME {ITEMA=VALUE ITEMB=VALUE ...}
;@ENUM ENUM_FOO
;{
; ITEMA=VALUE
; ITEMB
;}
```

Enum items can be used as any other macro, by their identifier.

- **If** – Controls compilation of portions of the script. If the expression written (after the directive identifier) evaluates as true, the code group following the directive is retained in the translation unit.

```
;#DEFINE DEBUGLEVEL 1
;#DEFINE FOO “STRING”
;#DEFINE BAR 4.5

;@IF DEBUGLEVEL > 1 && DEBUGLEVEL < 3
;{
;     PRINT “LOG LEVEL A: DEBUG MESSAGE”
;}

;@IF ((DEBUGLEVEL <= 12) || ((FOO == “STRING”) && FOO != 4.25))
;{
;     PRINT “LOG LEVEL X: DEBUG MESSAGE”
;     IF EVAL (OCTOPI.TENTACLES == “CSE > SKYRIM”)
;           PLAYER.KILL
;     ENDIF
;}
```

The condition expression can only include macro identifiers and constants/literals. The directive supports the following relational operators, which are evaluated in their default order of their precedence.
  - Equality [==]
  - Less than or equal [<=]
  - Greater than or equal [>=]
  - Inequality [!=]
  - Greater than [>]
  - Less than [<]

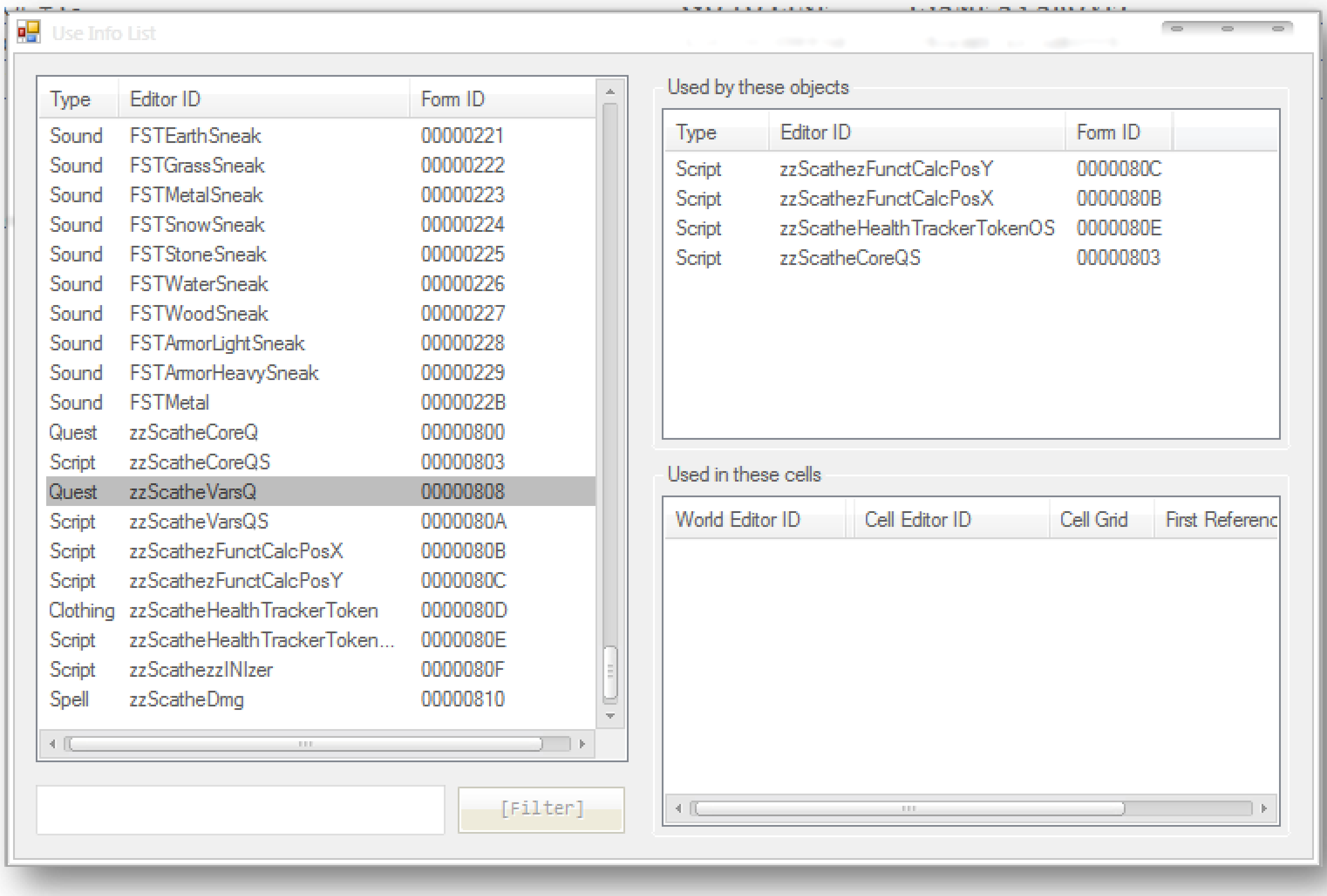
In addition to the above, the logical operators AND [&&] and OR [||] are allowed in expressions. Parentheses may be used to override the default precedence.

SHORTCUT KEYS AND COMBOS

The CSE Editor adds a number of counter-intuitive shortcut keys for its various functions, beyond what is supported by the common text edit field.

Shortcut Key	Action
MIDDLE MOUSE CLICK ON A TAB	**CLOSE TAB**
CONTROL + T	**NEW WORKSPACE**
CONTROL + TAB	**SWITCH BETWEEN WORKSPACES IN THE FORWARD DIRECTION**
CONTROL + SHIFT + TAB	**SWITCH BETWEEN WORKSPACES IN THE BACKWARD DIRECTION**
CONTROL + NEW BUTTON	**NEW WORKSPACES AND CREATES A NEW SCRIPT**
SHIFT + NEW BUTTON	**NEW TAB CONTAINER**
CONTROL + OPEN BUTTON	**NEW WORKSPACES AND INITIALIZE OPEN SCRIPT DIALOG**
CONTROL + Q	**TOGGLE COMMENT**
CONTROL + O	**OPEN SCRIPT**
CONTROL + S	**COMPILE AND SAVE SCRIPT**
CONTROL + D	**DELETE SCRIPT**
CONTROL + ALT + LEFT	**PREVIOUS SCRIPT**
CONTROL + ALT + RIGHT	**NEXT SCRIPT**
CONTROL + N	**NEW SCRIPT**
CONTROL + B	**TOGGLE BOOKMARK**
CONTROL + ENTER	**SHOW INTELLISENSE INTERFACE**
SHIFT + ENTER	**SUPPRESS LINEFEED**
ESCAPE	**HIDE INTELLISENSE INTERFACE**
HOME	**MOVE CARET TO LINE START**
CONTROL + F	**FIND (FOCUSES THE COMMON TEXT BOX, ENTER QUERY AND HIT ENTER)**
CONTROL + H	**REPLACE (SAME AS ABOVE)**
CONTROL + G	**GO TO LINE (SAME AS ABOVE)**
CONTROL + T	**GO TO OFFSET (SAME AS ABOVE)**
F1 (IN THE SELECT SCRIPT DIALOG)	**USE REPORT FOR THE SELECTED SCRIPT**

- **Centralized Use Info Listing**


Use Info List

Type	Editor ID	Form ID
Sound	FSTEarthSneak	00000221
Sound	FSTGrassSneak	00000222
Sound	FSTMetalSneak	00000223
Sound	FSTSnowSneak	00000224
Sound	FSTStoneSneak	00000225
Sound	FSTWaterSneak	00000226
Sound	FSTWoodSneak	00000227
Sound	FSTArmorLightSneak	00000228
Sound	FSTArmorHeavySneak	00000229
Sound	FSTMetal	0000022B
Quest	zzScatheCoreQ	00000800
Script	zzScatheCoreQS	00000803
Quest	zzScatheVarsQ	00000808
Script	zzScatheVarsQS	0000080A
Script	zzScathezFunctCalcPosX	0000080B
Script	zzScathezFunctCalcPosY	0000080C
Clothing	zzScatheHealthTrackerToken	0000080D
Script	zzScatheHealthTrackerToken...	0000080E
Script	zzScathezzINIzer	0000080F
Spell	zzScatheDmg	00000810

Used by these objects

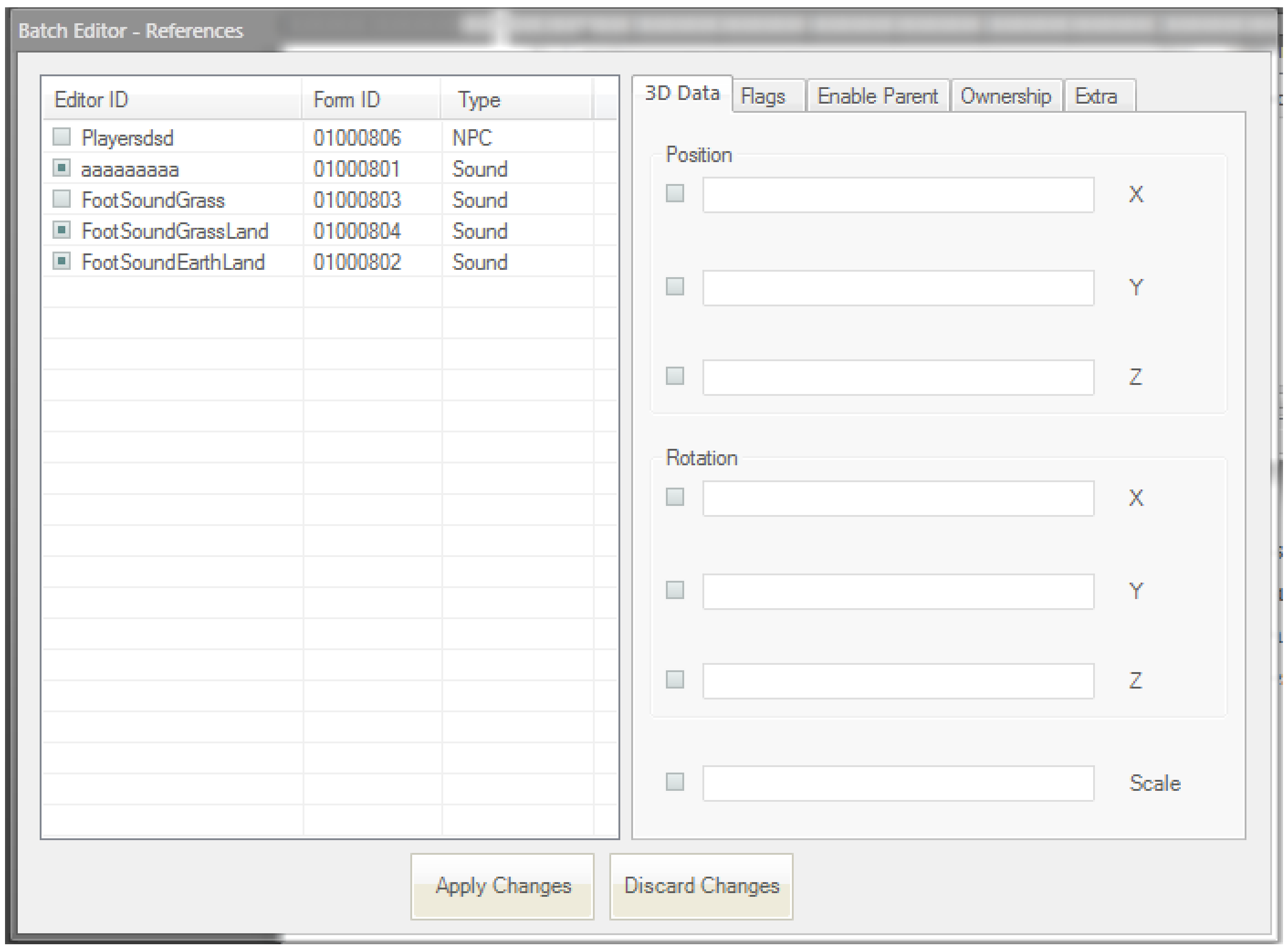
Type	Editor ID	Form ID
Script	zzScathezFunctCalcPosY	0000080C
Script	zzScathezFunctCalcPosX	0000080B
Script	zzScatheHealthTrackerTokenOS	0000080E
Script	zzScatheCoreQS	00000803

Used in these cells

World Editor ID	Cell Editor ID	Cell Grid	First Referenc

The use info listing tool is basically a conglomeration of the use reports of every loaded record in the CS. It allows easy look up of cell and object use lists through its centralized listing. Furthermore, every item in the list can be edited directly by double clicking it. The textbox at the bottom is used to filter the form list by editorID and formID. Every form type, save MGEF and GMST, are listed and tracked. The tool can be accessed from the main menu [Gameplay > Use Info Listings].

- **Batch Reference Editor**


Batch Editor - References

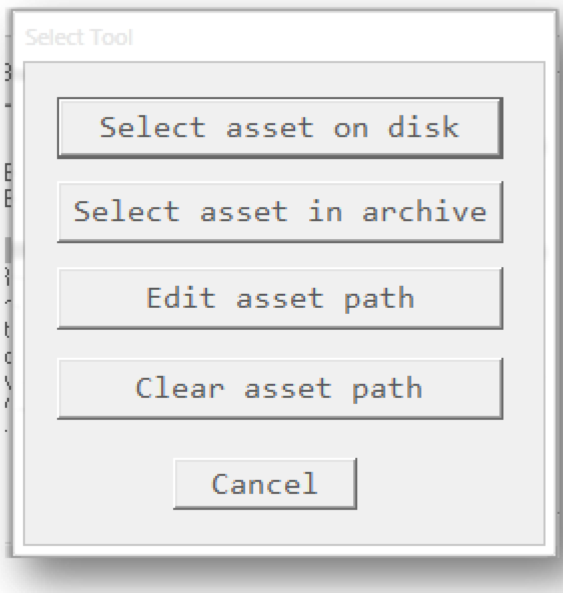
Editor ID	Form ID	Type
☐ Playersdsd	01000806	NPC
☑ aaaaaaaaa	01000801	Sound
☐ FootSoundGrass	01000803	Sound
☑ FootSoundGrassLand	01000804	Sound
☑ FootSoundEarthLand	01000802	Sound

The batch editor for references lives up to its name in most areas of the batch editing of references. It can either be invoked through the main menu [World > Batch Edit References] or the Render window's context menu. The editor will not initialize unless there is currently a cell loaded in the render window. Consequentially, editable references must be present in the loaded cell. When initialized, the selected objects will automatically be checked in the editor's object list. Only checked objects may be modified. The editor attempts to emulate the vanilla reference properties dialog as seen above: It can edit attributes of the following groups:

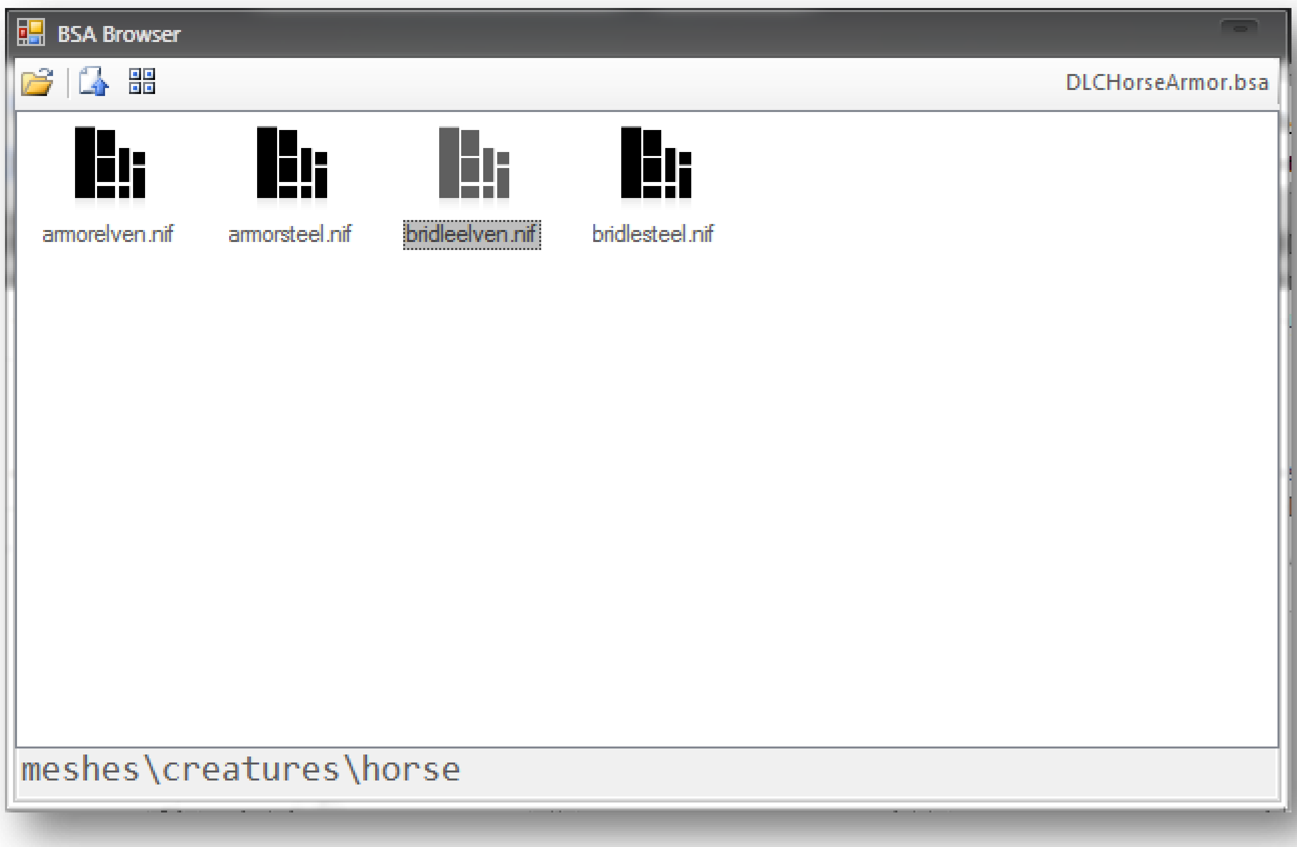
- 3D Data – Includes position, rotation and scale data.
- Flags – Includes flags for persistence, initially disabled state and visible when distant [VWD].
- Enable Parent, Ownership and Extra – Similar to the vanilla reference property dialogs.

The last 3 groups are applied selectively, to the objects that have the respective attributes and extradata. Each modifiable attribute has a checkbox next to it which indicates the enabled state of the corresponding attribute – The attributes modified are those with an 'Enable' check in their companion checkbox.

- **Enhanced Asset Selection Tool**

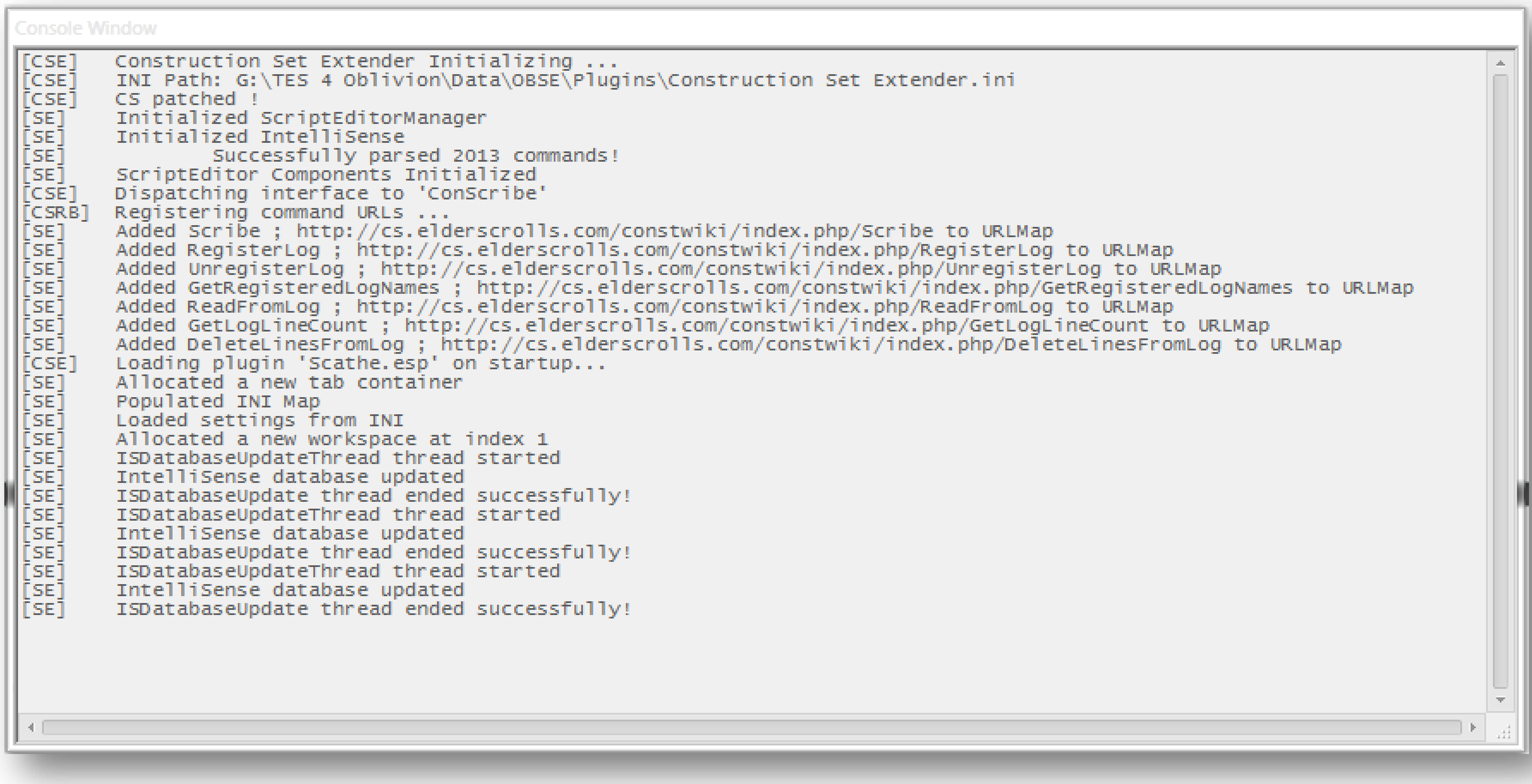


Asset selection i.e., selection of textures, meshes, sound files, speedtree files and animation files, has been overhauled for intuitive access. Clicking on an 'Add Asset ' button brings up the dialog on the left. The first button performs the vanilla operation of showing the 'Select File' screen. The second allows the selection of assets inside BSA archives, without having to extract them first. Selecting the second option brings up the BSA Browser dialog. The BSA browser is used to open TES4 BSA archives and select files from inside them. It automatically filters the BSA's file list by the required asset type. It supports Large Icon and List views.



The 3rd option lets the user directly modify the path of the asset file. 'File missing' warnings are suppressed during this operation. The 4th operation simply clears any existing path to the asset file.

- **Console Window**



```
[CSE]   Construction Set Extender Initializing ...
[CSE]   INI Path: G:\TES 4 Oblivion\Data\OBSE\Plugins\Construction Set Extender.ini
[CSE]   CS patched !
[SE]    Initialized ScriptEditorManager
[SE]    Initialized IntelliSense
[SE]            Successfully parsed 2013 commands!
[SE]    ScriptEditor Components Initialized
[CSE]   Dispatching interface to 'ConScribe'
[CSRB]  Registering command URLs ...
[SE]    Added Scribe ; http://cs.elderscrolls.com/constwiki/index.php/Scribe to URLMap
[SE]    Added RegisterLog ; http://cs.elderscrolls.com/constwiki/index.php/RegisterLog to URLMap
[SE]    Added UnregisterLog ; http://cs.elderscrolls.com/constwiki/index.php/UnregisterLog to URLMap
[SE]    Added GetRegisteredLogNames ; http://cs.elderscrolls.com/constwiki/index.php/GetRegisteredLogNames to URLMap
[SE]    Added ReadFromLog ; http://cs.elderscrolls.com/constwiki/index.php/ReadFromLog to URLMap
[SE]    Added GetLogLineCount ; http://cs.elderscrolls.com/constwiki/index.php/GetLogLineCount to URLMap
[SE]    Added DeleteLinesFromLog ; http://cs.elderscrolls.com/constwiki/index.php/DeleteLinesFromLog to URLMap
[CSE]   Loading plugin 'Scathe.esp' on startup...
[SE]    Allocated a new tab container
[SE]    Populated INI Map
[SE]    Loaded settings from INI
[SE]    Allocated a new workspace at index 1
[SE]    ISDatabaseUpdateThread thread started
[SE]    IntelliSense database updated
[SE]    ISDatabaseUpdate thread ended successfully!
[SE]    ISDatabaseUpdateThread thread started
[SE]    IntelliSense database updated
[SE]    ISDatabaseUpdate thread ended successfully!
[SE]    ISDatabaseUpdateThread thread started
[SE]    IntelliSense database updated
[SE]    ISDatabaseUpdate thread ended successfully!
```

The console window is the standard output for all of the construction set's (and CSE's) output operations. It logs messages from various components of the CS, giving each an identifiable prefix. Commands can be entered into the console through the textbox at the bottom of the window (not pictured in the above screenshot). 3rd party OBSE plugins may use the API to write their own output to the console and add custom commands. The window's visibility can be modified through the main menu [View > Console Window]. Default commands:

- `LoadPlugin string:<plugin name.extension> bool:SetAsActive` – Marks the parameter plugin as loaded and initiates plugin loading.
- `LoadForm string:<editorID>` – Opens the parameter form's dialog for editing. References will be loaded into the render window.
- `SavePlugin` – Saves the active plugin.
- `AutoSave` – Saves the active plugin to "Data\Backup\" as a copy.
- `Exit` – Closes the CS.

- **Quick-Load Plugin**

  The Data dialog has a new checkbox in it that calls itself 'Quick-Load Plugin'. When checked, the CS loads the active plugin exclusively. This feature is primarily useful when there is a need to look up a particular record in a plugin quickly. As plugin masters are skipped, the loading process my generate errors in case there are master-dependent records in the active file.

- **Startup Plugin**

  The other new control in the Data dialog, 'Set As Startup Plugin', is used to auto-load a plugin on CS startup. Clicking on the button causes the currently selected item in the plugin file list to be set as the startup plugin. Once set up, the plugin will be loaded as the active file on CS startup.

- **Load Script Window & Script on Startup**

  CSE comes with INI settings to allow the spawning of a script window on CS startup, as well as loading a script into it. These settings may be accessed through CSE's INI manager GUI. [File > CSE Preferences]

- **Hide Unmodified Forms**

  This tool toggles the visibility of records that aren't modified by the active plugin. It covers every list view control that is populated with record items. [View > Hide Unmodified Forms]

- **Save As**

  This tool allows the active plugin to be saved under a different name. [File > Save As]

- **Hide Deleted Forms**

  This tool toggles the visibility of records that have been marked for deletion. It covers every list view control that is populated with record items. [View > Hide Deleted Forms]

- **Set Form ID**

  This context menu tool allows the modification of the formID of the selected record.

- **Mark as Unmodified**

  This context menu tool marks the selected record as unmodified. Unmodified records aren't saved to the active plugin.

- **Undelete**

  This context menu tool restores deleted records. However, previous references to it aren't restored.

- **Jump to Central Use Info List**

  This context menu tool opens CSE's Use Info Listing tool and selects the selected record.

- **Unload Current Cell**

  Unloads the cell loaded into the render window. [World > Unload Current Cell]

- **Enhanced Response Editor**

  The response editor has been modified to provide a more streamlined interface to mod authors. The voice recording tool has been removed, given its obsoleteness in comparison to 3rd party recording tools such as Audacity. A 'Copy External File' tool has been added. It allows the user to move recorded voice files from arbitrary workspaces and into the CS's. It works on a per-race, per-sex basis – The target voice must be selected from the voiced races list in the editor.

  CSE also rids the necessity to switch between editor versions to generate LIP files for voices – It implements the lip sync generator in the latest version of the Construction Set. Lip files are generated on a per-race, per-sex basis, similar to the 'Copy External File' tool. The lip generator no longer needs a valid WAV file of the recorded voice for its working – It will automatically convert the source MP3 file to WAV during generation.

API

CSE comes with an API of its own that exposes some of its components to 3rd party OBSE plugins. Inter-plugin communication is done through the OBSE messaging system. To get a pointer to the 'interface object', plugins must pass a message of type 'CSEI' to the plugin 'CSE' during OBSE's PostPostLoad message callback; CSE registers listeners during the PostLoad callback. On receiving a 'CSEI' message from a plugin, CSE dispatches a message of the same type but with the data pointer linked to its interface object. The receiving plugin has to statically cast the opaque pointer to the interface's type. The interface is as follows, ready for immediate use:

```
#pragma once

/********** CSE Interface API ****************************************************
*     Interface object passed through the OBSE messaging system. A pointer to the
*     object will be dispatched to plugins that pass an arbitrary message of type
*     'CSEI' post post-plugin load (reply will be of the same type).
*
*     All other interfaces need to be initialized by calling InitializeInterface().
**********************************************************************************/

struct CSEInterface
{
      enum
      {
            kCSEInterface_Console = 0,
            kCSEInterface_IntelliSense
      };

      // Used to initialize CSE's interface objects. Similar to OBSE's QueryInterface.
      void*             (* InitializeInterface)(UInt8 InterfaceType);
};

struct CSEIntelliSenseInterface
{
      // Registers an arbitrary URL to a script command. Registered URLs will be displayed in the
      // script editor's context menu when the corresponding command is selected.
      void              (* RegisterCommandURL)(const char* CommandName, const char* URL);
};

struct CSEConsoleInterface
{
      typedef void      (* ConsolePrintCallback)(const char* Message, const char* Prefix);

      // Prints a message to the console. Prefix can be an abbreviated string of any kind.
      // Printed messages will be of the following format: [<Prefix>]\t\t<Message>
      // Mustn't be called inside a callback.
      //
      // Reserved Prefixes: CMD, CSE, SE, CS, BSAV, USE, BE
      void              (* PrintToConsole)(const char*      Prefix, const char* FormatString, ...);
      // Registers a handler that gets called whenever a message is printed to the console.
      void              (* RegisterCallback)(ConsolePrintCallback Handler);
};
```