Construction Set Extender

shadeMe

Version 6.0

# Table of Contents

***<add stuff>***

# Enhancements

# Plugins and Masters

##### 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 Data:

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. The behavior can be toggled through the “Save Options” submenu in the “File” menu.

##### Saving Plugins as ESM Files:

The CS can now save plugins as either ESP or ESM files.

##### “Save As” Tool:

Active plugins can be saved under a different name by using the new “Save As” tool, found in the File menu.

##### Timestamp Preservation:

The editor is now able to save plugins without modifying their “Last Modified” file timestamp. The behavior can be toggled through the “Save Options” submenu in the “File” menu.

# Workflow

##### Startup Options:

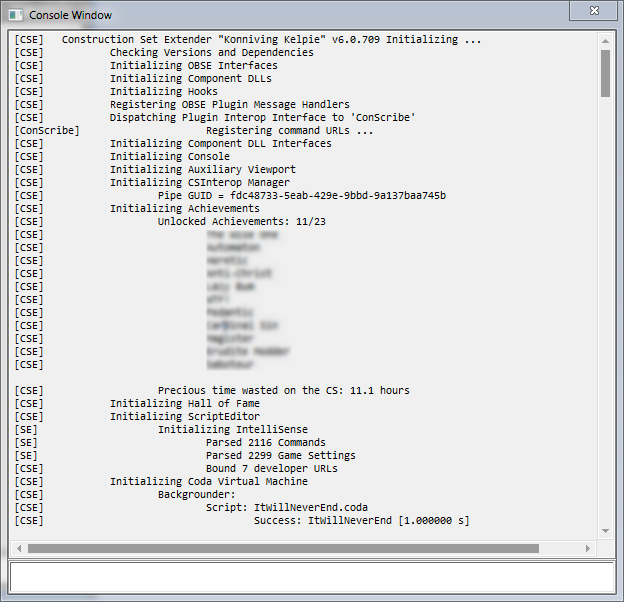
CSE allows the user to automatically perform the following operations right after the editor’s startup:

* + - Load a plugin and set it as the active file.
    - Load a script
    - Change the active workspace

The startup plugin can be set in the Data dialog, by selecting the required file in the listview and clicking on the ‘Set as Startup File’ button. The startup script/workspace can be set through the ‘CSE Preferences’ dialog, which is invoked from the File menu.

##### Console:

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. Its various tools can be accessed from its context menu, which can be brought up by right clicking anywhere in inside the window.



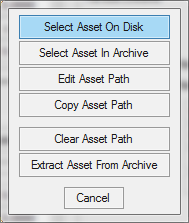
Certain commands can be entered and executed through the command line at the bottom of the window. Default commands:

* + *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.

As with the runtime, the up and down arrow keys can be used at the command line to walkthrough its command history.

##### Enhanced Asset Selection:

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 shown below.



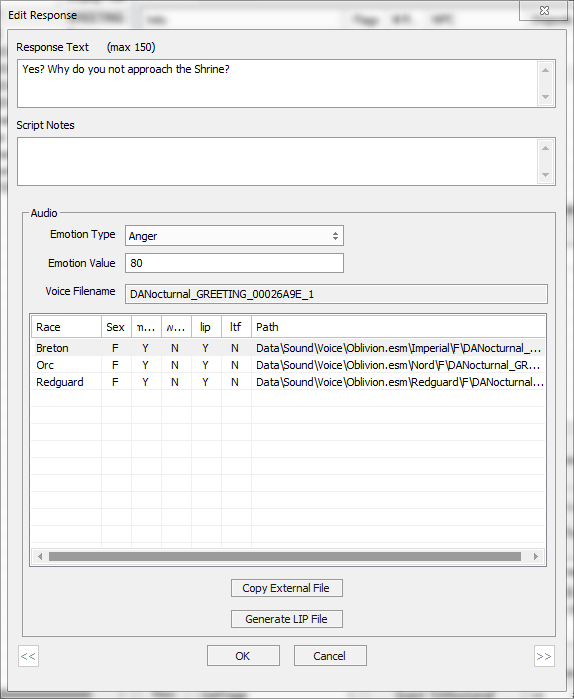
* + - * Select Asset On Disk – Opens the regular “Open File” dialog for disk access.
      * Select Asset In Archive – Opens the BSA browser, allowing the direct selection on assets inside BSA archives.  
          
        
      * Edit Asset Path – Allows the direct editing of the asset’s file path.
      * Copy Asset Path – Allows the quick copying of asset file paths between records.
      * Clear Asset Path – Resets the file path.
      * Extract Asset From Archive – Searches for the file inside any of the loaded archives and extracts it, if found.

##### Enhanced Find Text:

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.

##### 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, if any, to WAV during generation.

CSE also adds a batch LIP generator tool that generates lip files for all loaded topics. It can be accessed from the “Character” menu.

##### Form Enumeration Filters:

The “Hide Unmodified Records” and “Hide Deleted Records” tools can be used to toggle the display state of forms that haven’t been modified by the active plugin or have been deleted, respectively. They can be accessed from the “View” menu.

##### Global Script Creation:

This tool allows quest scripts to be quickly created by specifying the editorIDs of both the quest and its script, along with the processing delay time. It can be accessed from the Gameplay menu.

##### Context Menu Tools:

CSE adds a number of new tools that can be accessed from any form’s context menu:

* Set FormID – Allows the formID of a form to be changed.
* Mark As Unmodified – Reverts the “Modified” flag on a form, preventing it from being saved to the active plugin.
* Undelete – Resets the “Deleted” flag on a form.
* Show Override List – Displays a list of all the loaded plugins that modify the form in question.
* Edit Base Form – Only visible for object references. Opens the reference’s base form edit dialog.
* Toggle Visibility – Only visible for object references. Toggles the visibility state of the reference.
* Toggle Children Visibility – Only visible for object references. Toggles the visibility state of the reference’s linked children.
* Preview – View the form’s 3D representation in the Preview window.
* Jump To Central Use Info List – Displays the central use info list window and scrolls to the form in question.
* Add To Active Tag – Adds the form to the Tag Browser’s selected tag, if any.

##### Batch Copy Eyes/Hair:

Hair and eye records of one race can be copied to another by using the “Copy Hair” or “Copy Eyes” tool found in the Face Data tab of the Race edit dialog.

##### Launch Game:

The “Launch Game” button, found on the main toolbar, is primarily used to spawn a mad, wild killer bull – cunningly disguised as a bird – that’s got winning odds of 80000 to 1 in cock fight.

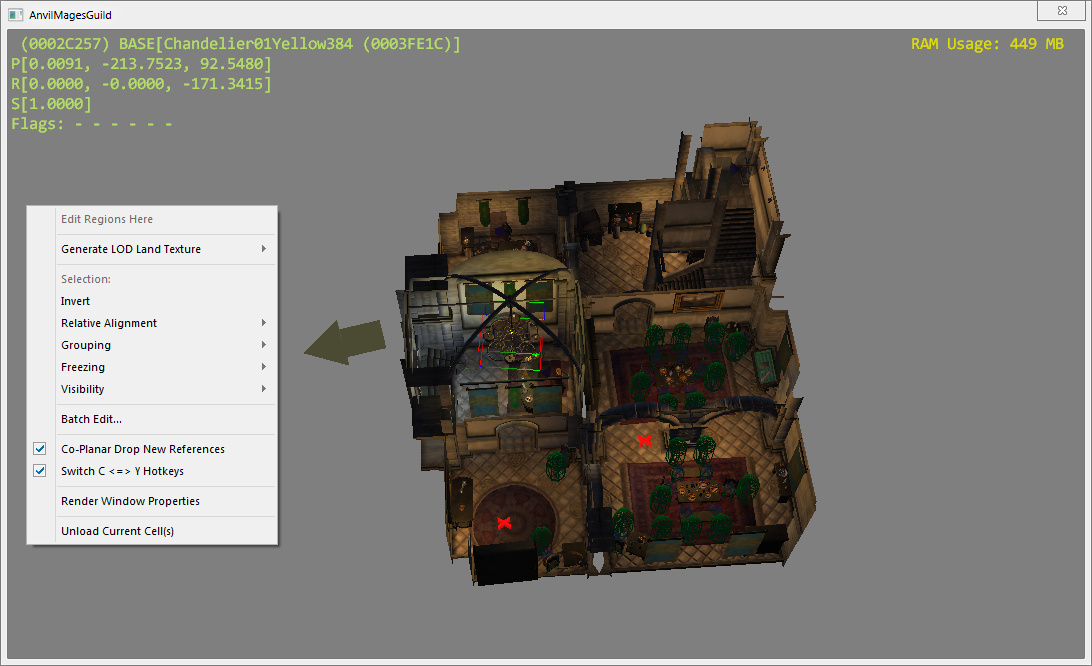
##### Workspaces:

CSE allows the user to switch between multiple working directories when using the editor. Each workspace can be considered a separate root directory (one that contains the “Data” folder) that can house plugins and asset files independent of each other. Custom workspaces need to be placed inside the original game directory. Master files present in the default workspace (*<root>\Data*) are shared with custom workspaces.

The “Set Workspace” tool can be accessed from the File menu.

##### Render Window:

The following improvements have been made to the render window:



* Increased Responsiveness – The viewport now operates at an average framerate of 48 FPS.
* On-screen Display - Detailed information on the active selection and the editor’s current RAM usage are displayed in the top corners of the window.
* Door markers Properties - Holding down the Control key and double clicking on door markers will now bring up their reference properties dialog box.
* New Context Menu Tools –
  + Invert – Inverts the render window selection.
  + Relative Alignment – Aligns the selection to one of the 3 axes of its first reference.
  + Grouping - Allows multiple references to be grouped as a collective object. Selecting a member of a group using the left mouse button causes all of its members to be selected. This behavior will persist until the selection group is dissolved. A reference can be a member of only one group at a time.
  + Freezing - Freezes the movement of the selection. Frozen references cannot be moved in any way until they are thawed.
    - Thaw All In Cell – Un-freezes all frozen references in the loaded cell.
    - Freeze All Inactive – When enabled, all unmodified references are implicitly frozen to prevent accidental edits.
  + Visibility – Toggles the visibility of the selection/selection’s linked (enable parent) children.
    - Reveal All In Cell – Resets the visibility state of all the references in the loaded cell.
  + Batch Edit – Displays the batch reference editor for the loaded cell.
  + Co-Planar Drop New References – When enabled, new references will be placed co-planar with the object at the cursor location.
  + Switch C & Y Hotkeys – When enabled, the functionalities of the C (Isometric view) and Y (Axis modifier) is switched.
  + Unload Current Cell(s) – Unloads the cell(s) loaded into the render window.

New Hotkeys:

|  |  |
| --- | --- |
| Shortcut Key | Action |
| F1 | Show use info report for selection |
| 2 | Toggle selection visibility |
| SHIFT + V | Invert selection |
| SHIFT + E | Toggle co-planar dropping |
| SHIFT + Q | Freeze/Release auxiliary viewport camera |
| CONTROL + Q | Switch between the vanilla and alternate movement settings |
| CONTROL + R | Unlink path grid points from their linked references |

##### Script Compiler:

The following improvements have been made to the script compiler:

* Compiled bytecode size for scripts has been increased to 32KB.
* Compiler errors accumulate, i.e., script compilation will not halt on encountering an error.

##### LOD Texture Generator:

The following improvements have been made to the LOD texture generator:

* Diffuse map and normal map textures are created with the appropriate mipmap chains.
* The resolution of diffuse maps can be customized through the INI file, with the new upper-limit being 6144px.

# User Interface

##### Safer Modification of List View based Records:

Forms of types such as Magic Effect, Race, Eyes, Hair, etc. (with the lone exception of Quests) are displayed as a list when editing them in the CS. The default behavior of such dialogs leads to many a dirty edit. CSE attempts to fix it by introducing the following changes:

* The ‘OK’ and ‘Cancel’ buttons are changed to ‘Apply’ and ‘Close’ respectively, thereby clarifying the actions they perform, i.e., ‘Apply’ saves any changes made to the currently selected record while ‘Close’ discards them and closes the edit dialog.
* The newly minted ‘Apply’ button no longer closes the dialog, thereby allowing the user to continue editing after saving.
* When switching to a different record, a ‘Save Changes’ confirmation is displayed. Changes are saved only when the user selects ‘Yes’.

# Trifles

##### Fast Exit:

The editor shuts down in matter of seconds, as opposed to minutes when not using the CSE.

##### Icons with Mipmaps/Texture Size Limitations:

Icons with mipmaps can be previewed correctly and the CS no longer generates errors about the matter. Also, the resolution limitation of 512px for certain textures has been removed.

##### Auto-loading BSA Archives:

All BSA archives in the Data folder are loaded at startup, regardless of their connection to an active plugin.

##### Integer-prefixed EditorIDs:

EditorIDs that start with integers optionally (i.e., as controlled by the appropriate INI setting) display a warning, reminding the user of the caveats of using such identifiers.

##### Integer-prefixed EditorIDs:

EditorIDs that start with integers optionally (i.e., as controlled by the appropriate INI setting) display a warning, reminding the user of the caveats of using such identifiers.

# Vanilla Bugfixes

The CSE fixes a ton of bugs and quirks present in the vanilla CS. The complete list follows:

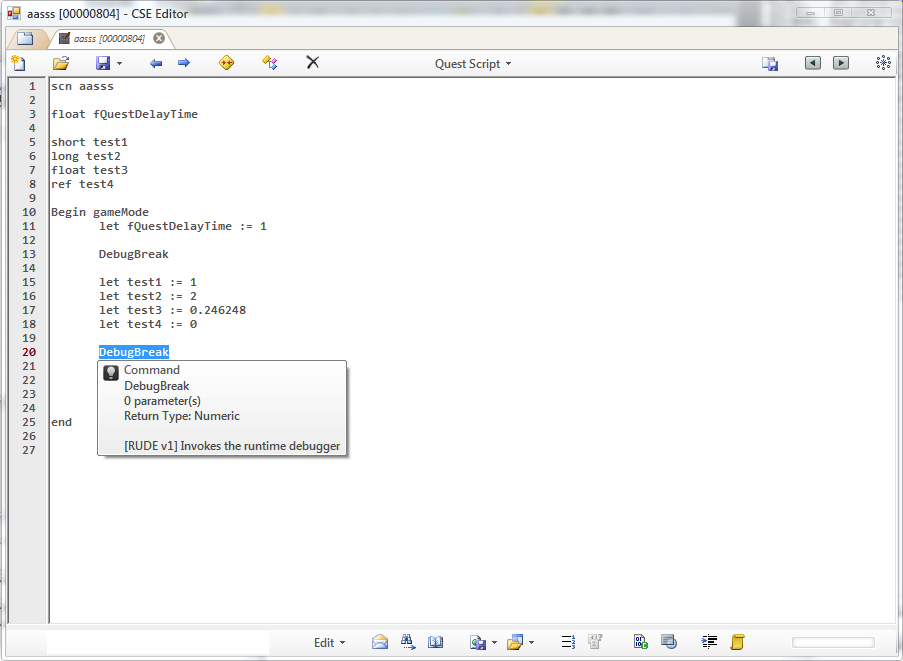
|  |  |
| --- | --- |
| Response Editor Microphone | Fix for the CTD that occurs on machines with Realtek soundcards, when the response window is initialized without a microphone plugged in its socket. |
| Topic Info Data Reset | Fix for the bug that automatically clears result script text and info flags if they are visible when a new topic is added to the topic list. |
| NPC Editor Face-Gen | 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 Edits | 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-Edits | Fix for the bug that copies race description from one race record to another if the latter were to invoke the spell checker. |
| Code Level Assertions | This bug is deep rooted in the editor code and tends to cause a fairly large number of CTDs for no reason. |
| Unknown Record And Group Types In Plugins | Fix for the bug that caused a CTD when a plugin containing unknown records, sub records or group types was loaded into the editor. |
| Plugins With Missing Masters | Fix for the bug that caused the editor to exit when a loaded plugin was found to have a missing master. |
| Always-On-Top Data Dialog | Fix for the bug that caused the Data dialog to stay above all open windows. |
| Render Window Main Menu Item | Fix for the bug that prevented the Render window for being closed when using the View > Render Window main menu item. |
| Topic Info Copying | Fix for the bug that caused the wrong topic info record to be flagged as modified during a copy operation. |
| Lip Sync File Generator | The infamous lip sync tool has been finally fixed! More details in the next section. |
| Variable Declarations In Result Scripts | Fix for the bug that caused a CTD when compiling a result script that had local variable declarations. |
| New/Duplicate Form Selection | Fix for the bug that prevented newly created/duplicated list view-based forms from being selected after instantiation. |
| Reference Duplication | Fix for the bug that prevented the complete duplication of extradata between object references. |
| Plugin Author/Description Editing | Editing a plugin’s author/description field in the Data dialog sometimes corrupts it, the cause of which appears to be related to file access permissions. CSE attempts to prevent this by preemptively check the plugin file’s handles before committing such potentially dangerous changes. |
| Reference Record Serialization | Fix for the bug that prevented the correct serialization of deleted reference records under certain conditions. |
| Game Setting Cleanup | Fix for the limitation that caused modified game setting records to retain their state between plugin loads. |
| Form Usage Reference Counting | Fix for the bug that broke usage reference counting when a form was referenced more than once by another. |
| Anim-Object Creation | Fix for the bug that caused a CTD when an Anim-Object was created/edited when no plugins were loaded. |
| LOD Diffuse Map Filename | Fix for the bug that caused incorrect file names to be given to generated LOD color maps (which led to pink colored LOD meshes in-game). |
| Plugin Save | Fix for the bug that prevented the “Save Plugin” tool from being reused if a previous operation ended prematurely. |
| Incorrect Script Compiler Errors | Fix for the bug that displayed incorrect line numbers in script compiler error messages. |
| Light Object Reference First-Time Initialization | Fix for the bug that caused a CTD when a light object was placed in a cell for the first time. |
| Render Window “Fall” Operation | Fix for the bug that caused occasional CTDs when using the “Fall” tool. |
| Quest Stage Result Scripts | Fix for the bug that prevented quest stage result scripts from being cleared. |
| Reference Editor Z-Order | Fix for the bug occasionally caused reference edit dialogs to spawn underneath the render window. |
| NPC Editor Inventory Preview | Fix for the bug prevented NPC models from being updated correctly in the preview window after an item was removed from their inventory. |
| Faction Ownership In Conditions | Fix for the bug that prevented factions from being allowed as arguments to condition script commands that accepted parameters of the type “Owner”. |
| ESP/ESM File Associations | Fix for the bug that trashed ESP/ESM file associations at editor startup. |
| Path Grid Point Linking | Fix for the bug that invalidated the render window after a path grid point was linked to a reference. |
| Cell View Window Bounds | Fix for the bug that progressively reduced the size of the cell view window’s controls every time plugins were loaded into the editor. |
| Theme-Enabled Owner-Drawn Controls | Fix for the bug that caused CTDs when painting theme-enabled owner-drawn preview controls. |
| Landscape Texture Change | Fix for the bug that caused the render window to lose input focus after switching the active landscape texture. |
| Path Grid Point Creation | Fix for the bug that caused a CTD when a path grid point was created after every loaded path grid was destroyed. |
| Path Grid Point Selection | Fix for the bug that prevented the selection rectangle from showing on multiple path grid point selections. |
| Render Window Reference Duplication | Fix for the bug that reverted the render window’s selection to the original references during a duplication operation. |
| Non-Standard Line Endings In Scripts | Fix for the bug that prevented the correct parsing of script source code containing non-standard line endings. |
| Dialog Creation Failure | Fix for the bug that prevented modeless dialogs from being destroyed correctly, eventually exhausting the OS”s global window handle pool. |
| LOD “Black Texture” | Fix for the bug that caused half the pixels of generated LOD diffuse maps to appear black in color. |
| Cell Edit Dialog Dirty Edits | Fix for the bug that caused dirty editors when selecting cells in the Cell edit window. |
| Superfluous Addition Of Cell Water Data | Fix for the bug that unnecessarily added water extradata to cells that didn’t have any water. |
| Render Window Exterior Cell Loading | Fix for the bug that caused the render window to flicker intermittently when loading exterior cells. |
| Interior Cell Duplication | Fix for the bug that prevented lighting data from being copied when an interior cell was duplicated. |
| Sweeping Path Grid Point Selection | Fix for the bug that caused the render window to select every path grid point in the loaded cell(s) while reclaiming input focus. |
| Quest Stage Log Entry | Fix for the bug that caused the quest editor to close whenever the Enter key was pressed inside the Log Entry textbox. |
| “Path Grid Edit Mode” Toolbar/Menu Deactivation | Fix for the bug that caused the “Path Grid Edit Mode” main menu/toolbar buttons to relinquish their toggled state occasionally for no reason. |
| Path Grid Edit Mode Initialization | Fix for the bug that caused path grid points to incorrectly appear preselected when entering the path grid edit mode. |
| Loading Detached References | Fix for the bug that caused a CTD when a detached reference (a reference with no parent cell) was loaded into the render window. |
| Reference Variables In Compound Expressions | Fix for the bug that caused a CTD when a reference variable was used as a calling reference in a compound script expression. |
| Mismatching Parentheses In Set Expressions | Fix for the bug that prevented the script compiler from complaining about mismatching parentheses in Set expressions. |
| “Dirty Flag” Reset | Fix for the bug that caused the editor to reset its “Unsaved changes” flag when the Data dialog was closed. |
| Weather Sounds Listview Sorting | Fix for the bug that caused a CTD when the sounds listview in the weather edit dialog was sorted. |
| Render Window Axis-Modifier Hotkeys | Fix for the bug that caused the render window’s axis modifier keys (Z, X and Y) to work incorrectly. |

# 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.



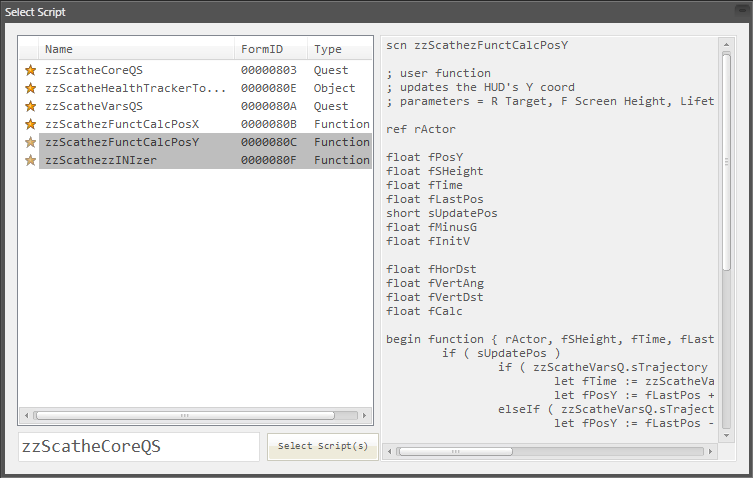
The CSE Editor is a tabbed script editor, as obvious as it may seem. Each editor window is called a tab container while the actual editor component (the one with the text editor control) is called a workspace. A tab container is always spawned with an empty workspace (think of a workspace as a vanilla 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:



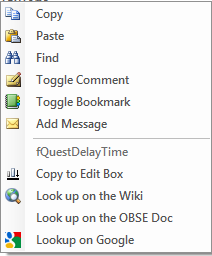
The list view displays all the loaded scripts and the text box to the right shows a preview of the selected script. The textbox at the bottom can be used to select a particular script by its editorID or formID. Multiple scripts can be selected in the list view during an ‘Open’ operation and this is where 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. Compilation results are logged to 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** – Accepts 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. All operations are case-insensitive.
  + ***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 an item will either move the caret to the appropriate line or remove it from the list depending upon the message type.
* **Find results** – Displays the last executed Find operation’s results. 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** – Enumerates every variable in the script, their type and index. Indices can be edited by double clicking on the desired cell.
* **Update variable indices** – Only used in conjunction with the above tool. Updates the script with the modified variable indices, if any. 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 the result in a separate text viewer.
* **Sanitize Script Text** – Performs various operations, as set in the preferences window, on the script text to make it more legible.
* **Bind Script** – Allows the active script to be bound to a new or existing scriptable object.
* **Progress bar** – Shows 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 yet further 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.
* **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.

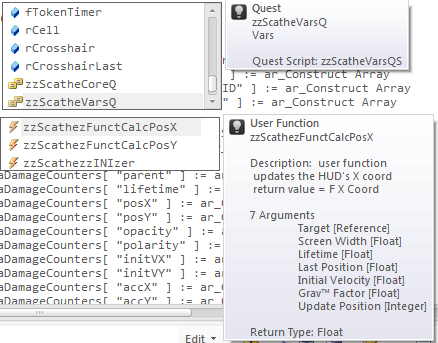
Beyond the above, the editor has 3 main components to it:

### **IntelliSense**

### **Script Validator**

### **A preprocessing engine called ‘A preprocessing engine’**

IntelliSense

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.
* 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 uses 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 support no more than one 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-numeric characters 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

* **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\Scripts\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 (i.e., C++’s) 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.

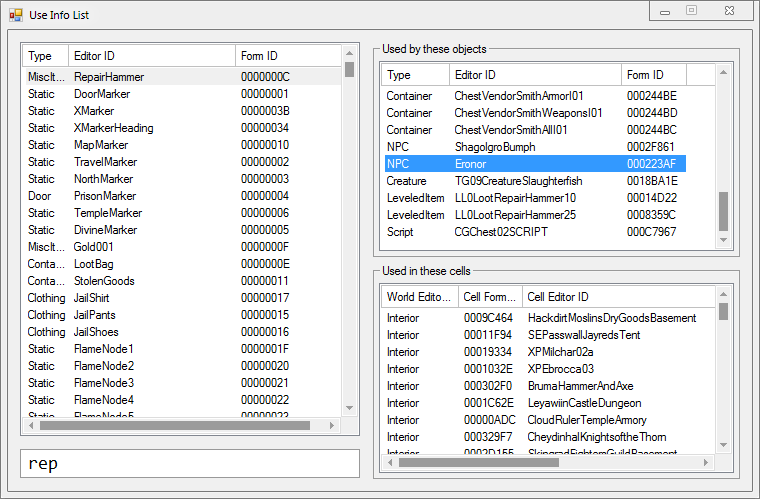
Standard directives can be defined in the ‘STDPreprocDefs.txt’ file. These are always parsed during each preprocessor operation. Multi-line blocks can contain directive declarations – They will be expanded automatically when the parent directive is.

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 | Opens a New Workspace And Creates A New Script In It |
| Shift + New Button | New Tab Container |
| Control + Open Button | Opens a New Workspace and Initializes the 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 |
| Escape | Hide IntelliSense Interface  Clear Find Result Indicators |
| 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 + E | Go to Offset (Same as above) |
| F1 (In the Select Script dialog) | Use Report for the selected script |

##### Centralized Use Info Listing:



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.

It can be accessed from the “View” menu.

##### Batch Reference Editor:



The batch editor for references lives up to its name in most areas of the batch editing of references. 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 will 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.

It can be accessed from the render window’s context menu.

##### Tag Browser:



The Tag Browser allows the user to attach arbitrary tags to any record. Tags can be nested with the use of drag-drop operations, renamed by (slowly) double clicking on their nodes. A record can be allocated multiple tags, but each tag may only contain a single instance. Records can be drag-dropped into the record list of the active tag. The record list behaves similar to the object window – Records can be double clicked for editing and drag-dropped into the render window for reference instantiation or indeed any other target location that allows drag-dropping. The textbox can be used to search for specific items in a tag’s record list. Tags/tagged records may be added/removed through the context menu. Tag hierarchies can be saved to disk using the ‘Save’ tool; the ‘Load’ tool is subsequently used to load a saved hierarchy. Invalid/non-existent records will be removed during a load operation. The textbox in the bottom can be used to find items in the record list.

It can be accessed from the “View” menu.