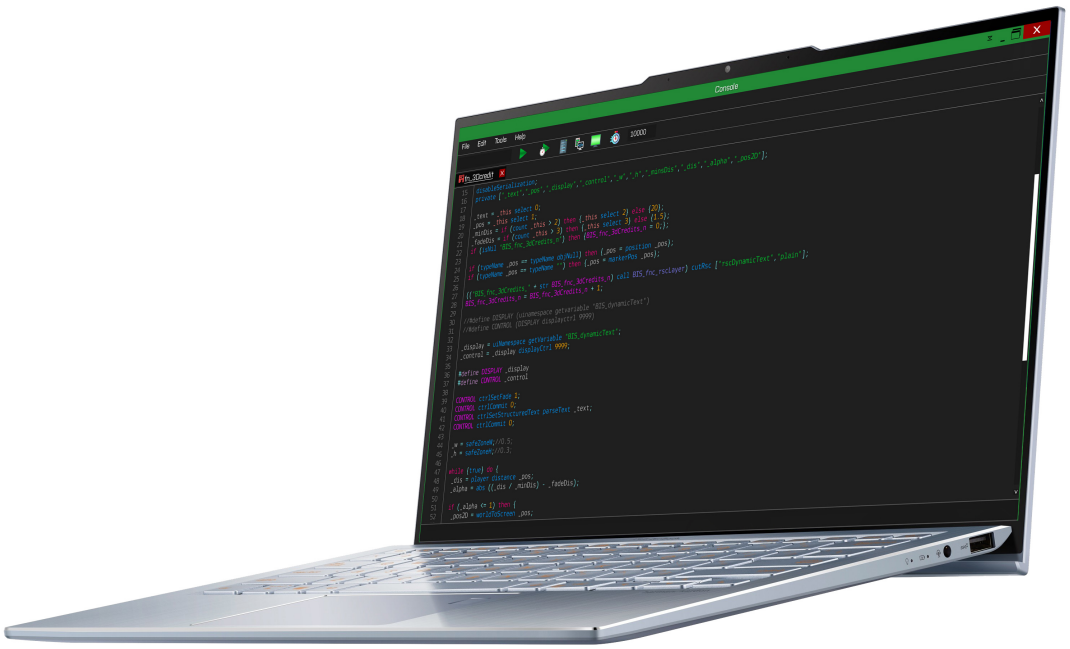


Advanced Developer Tools

By

Leopard20

Contents

1. Introduction

Advanced Developer Tools is an Arma 3 mod that adds several new and revamped developer tools to the game, namely the debug console, config viewer, function viewer, plus a few extra others.

As someone who has been creating mods for Arma for a long time, I found the vanilla and most existing mods insufficient for my needs, so I decided to create this mod to increase the productivity speed for the mod makers.

This documentation should explain everything you need to know to use this mod. If you find the document too long, you can start from the contents page and only read the parts that you're not familiar with.

2. Feature Overview

General Design:

- A familiar **Windows design**, with **resizable and moveable windows**. In addition to other Windows features like snapping the Window to the screen borders. Enjoy a multi window experience in Arma 3!
- **Customize the mod** the way you want. The mod offers a wide variety of customization options, including **theming**! You can recolor almost everything!

Debug Console:

- Enjoy SQF programming with **syntax highlighting**, with several **predefined themes**! (such as VSCode Dark+)
- **Multi-Tab design**! Edit multiple scripts at the same time!
- **Line numbers** for easier tracking
- **Auto-completion list** (similar to VSCode/Notepad++), including event handler names, functions, magic words, etc.
- Supports **preprocessor commands** (#define, #ifdef, #include, etc.) and line numbering for improved debugging inside the debug console.
- Improved code editing features, such as **Tabs (tab key), Case Conversion (Ctrl+Shift+U), Undo/Redo (Ctrl+Z, Ctrl+ Y), Word Delete (Ctrl+Backspace/Delete), Commenting Out (Ctrl + Q)**, and many many more!
- **Search and Replace** with various search options, along with **Normal, Extended, and Regex** search modes!
- View the **command syntax information** in a new and improved Scripting Help window (F1)
- Added a new button that lets you run the code in **Scheduled Environment** directly (no need for spawn anymore)
- Added a new button for **custom RemoteExec**: remoteExec the code for your target client(s)
- Added a text box for **providing arguments** to your test code (no need for _args call {...} anymore)
- **Open files** (from addons) and functions to view/edit them
- Dedicated **Execution History** window. Want to re-execute a command you tried some time ago? You know where to find it!
- An advanced **Color Picker** at your disposal, which can come really handy if you're designing GUI elements.

- **Magic Word**: Typing this into most text boxes will open the debug console, and you can continue editing your code in there, and save it back into that text box!

Watch:

- Define as many **Watched Expressions** as you want!
- **Customizable colors**
- **Pin** watched expressions so that you can see them during gameplay!

Draw:

- Define **custom "drawIcon"/"drawLine"s** with just a few clicks! And as many of them as you want!
- Supports AGL and ASL position formats
- Precompiled code for faster execution (despite the extensive available options)

Config Viewer:

- A **super-fast config viewer**. No more annoyingly long loading times when you open the Config Viewer!
- Custom designed tree view with “smart rendering” for faster performance (unlike vanilla RscTree)
- Customize the **config viewer colors** to your liking
- **Replaces the vanilla config viewer** in the context menu of Eden (right click on entity) for faster entity lookup.
- **Config Search**: Search in the config in classes,
- **Go to Config,** which supports both direct config paths and expressions
- Accidentally went to another config? Want to go back? Use the **Back and Forward** feature!
- An advanced **Data Viewer** for viewing various types of config data, including: arrays, text, images, colors, models, and sounds (sounds don't work very well at the moment)
- Dedicated **Bookmarks** window

Function Viewer:

- Supports **syntax highlighting**
- **Searching** in all available functions defined in mission config and config file
- **Recompile** functions by tag, category, or even individually.

3. Windows design

3.1. Moving and resizing

All windows are draggable, and some can even be resized (those with a maximize button, by default) and minimized. You can make all windows resizable by enabling the option in mod settings, but since they were not designed to be resizable you may encounter a lot of bugs.

You can resize windows in three ways, just like Windows:

1. Manual resize: drag the corners of the window
2. Snap resize: drag the window and drop in onto the edges of the screen
3. Fullscreen toggle: double click the window title, or use the dedicated fullscreen button.

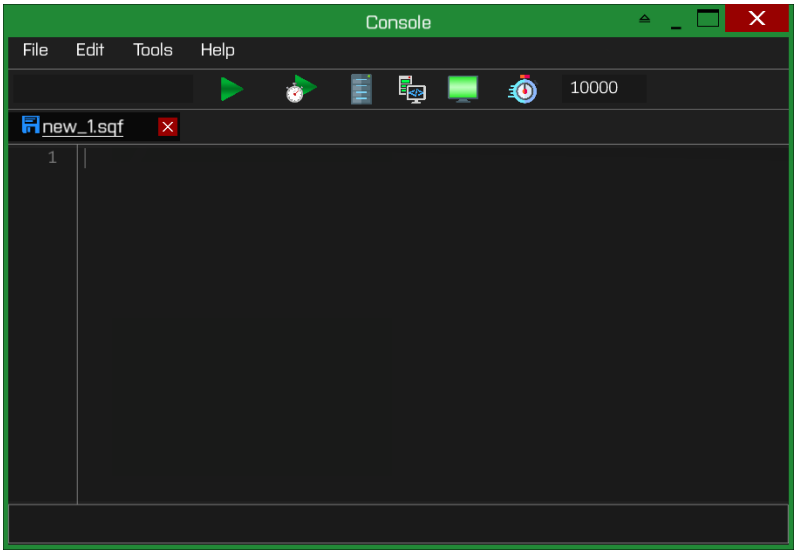


Figure 1 - The debug console window. Notice the standard Windows buttons at the top right.

3.2. Docked mode

The debug console has an additional window button called “Toggle docked mode”, which looks like ⏏. Docked mode puts the debug console directly above and aligned with the Watch tool, and prevents the debug console from moving.

Note that the **Config viewer, function viewer, animation viewer**, etc. have been moved under the Tools menu.

The height and width of the debug console in docked mode can be changed in settings.

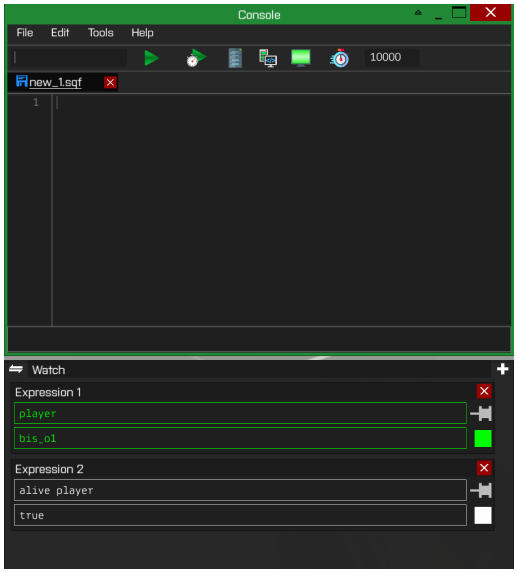


Figure 2 - Debug console in Docked mode

4. Debug console

4.1. Editing

The editing features are very similar to standard IDEs and code editors.

Multi-tab editing

It is possible to edit multiple documents at the same time using the multi-tab feature. You can open a new tab using the Control N (Ctrl+N) shortcut or through the File menu. To close a tab, use Control W (Ctrl+W).

Syntax highlighting

The contents of the debug console are automatically syntax-highlighted. The mod extracts all available commands in the game, so there's no need to update the mod to get the new commands that are introduced with game updates. However, things such as Magic Words, Event Handler and Preprocessor Commands are added manually.

Autocompletion

As you type, the mod will suggest new words to you based on commands, functions, and variable names defined in the current document. You can make a selection using the Tab (Tab) key.

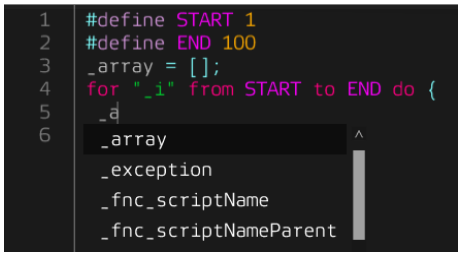


Figure 3 - Autocompletion list and syntax highlighting

Function parameter hints

The function parameter hint shows parameters as well as the return data type.

This is very important because sometimes the alternative syntaxes for some commands may return completely different data types (e.g. one of `createUnit` syntaxes doesn't return anything).

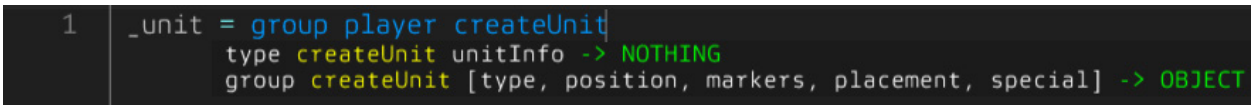


Figure 4 - Function parameter hints. Notice the return type in green

Command information

The command info window can be opened by pressing F1 (F1) while the caret is on a command.

New features compared to the vanilla command info:

1. Shows the **return types and parameter types** are shown directly under every parameter and return value in GREEN for easier visibility.
2. **Syntax highlighted** examples, and example return values are shown as comments.
3. You can cycle through all **alternative syntaxes** for a command.

4. **Link to wiki page** for the commands.
5. Different **categories**, such as scripting commands, event handlers, functions, magic variables, and preprocessing commands.

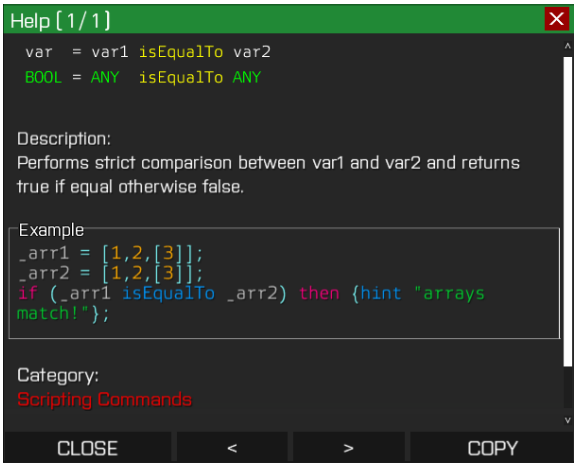


Figure 5 - Command info. Notice the highlighted example and command return types and parameter types in green

Tabs

You can insert tabs using the Tab (Tab) key. This will place a new tab at the start of the line, no matter where the caret is. The mod will keep the current indentation level (number of tabs) when you go to the next line.

Note that the tabs are only 1 character wide and hard to distinguish from spaces.

You can delete the tab using Shift Tab (Shift+Tab). Deleting tabs across multiple lines is also possible.

Case conversion

You can convert to uppercase using Control Shift U (Shift+Ctrl+U), and convert to lowercase using Control U (Ctrl+U) shortcuts. This feature is also available under the Edit menu.

Commenting out

You can comment out a line using the Control Q (Ctrl+Q) shortcut. If you haven't selected any characters, the current line (where the caret is) will be commented out using // (single line comment). If you select multiple lines/characters, the selection will be commented out using /* */ (multi-line comment)

You can uncommend a line using Control Shift Q (Ctrl+Shift+Q).

Underlining (highlighting)

If you select a word, all instances of that word in the current document will be underlined, which is the equivalent of highlighting in standard IDEs and editors (multiple highlights are not possible in Arma).

```
1  params ["_unit"];
2
3  if (alive _unit && lifeState _unit != "UNCONSCIOUS") then {
4   _unit setDamage 0;
5  }
```

Figure 6 - All instances of the word "_unit" are underlined when one is highlighted.

Search and replace

You can open the Search and replace window using Control F (Ctrl+F). Three search modes are available:

1. **Normal**: All characters are treated as they are.
2. **Extended**: All instances of **\n** (next-line), **\r** (carriage return) and **\t** (tab) in *Find what* and *Replace with* fields will be replaced with the appropriate character.
3. **Regex**: Regular expression search. For example, you can search for any word by searching \w+.

Some notes about regex:

Note that regex search doesn't support searching backward. Also, the option *Match whole word* doesn't work, and if needed it has to be provided using the appropriate regular expression. However, the *Match case* option works.

In addition, note that the mod uses the default C++ regex library. In that library, the **capture groups are represented in the format: $1, $2, etc.**, not \1, \2, etc. (what is used by Notepad++)

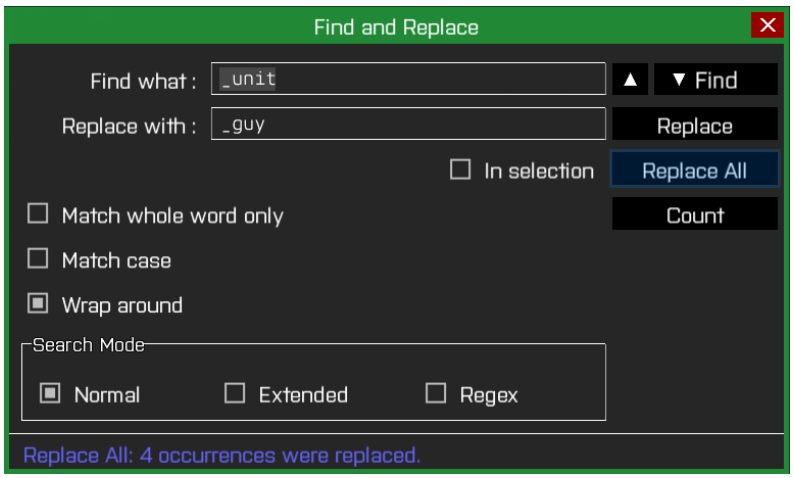


Figure 7 - Search and replace

Undo and redo

The editor supports undo and redo, so don't worry about making mistakes. You can **undo** using Control Z (Ctrl+Z) and **redo** using Control Y (Ctrl+Y) or Control Shift Z (Ctrl+Shift+Z).

Magic word

The mod has a feature called Magic Word, which allows you to open the debug console and edit many scripting fields (e.g. object init fields, trigger statements, etc.) simply by typing the magic word into them.

In addition, if you mistakenly disable the debug console and can't open it anymore, you can type the magic word into the vanilla debug console to make the mod's debug console reappear.

The default magic word is **DBUG** (note: no *E*!), but you can change the magic word in settings.

Note that the magic word only works in CT_EDIT controls that inherit from RscEdit or ctrlEdit.

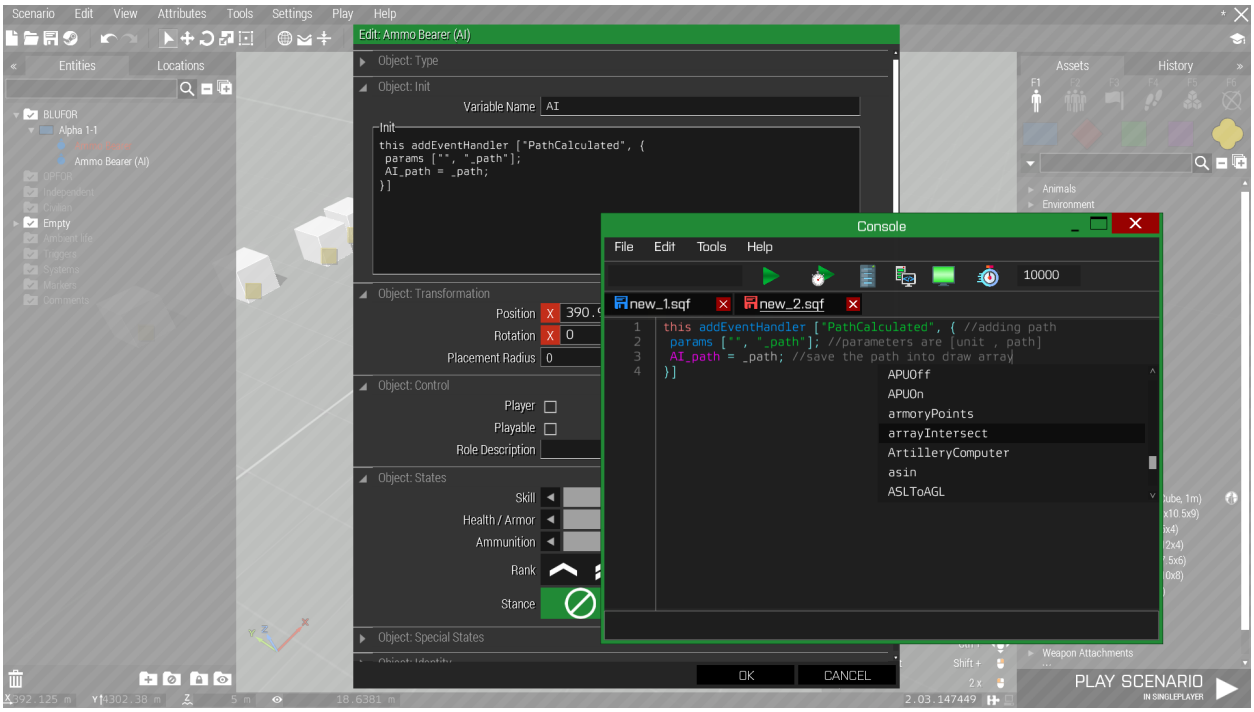


Figure 8 - Using the magic word to edit the init field of a unit in 3den. Note that the contents saved into the init are preprocessed.

Saving the document

At the time of writing, you can't save the document to an external file. However, the contents of the debug console (including all tabs) will be saved in the profileNamespace so you can safely close the debug console at any time you want.

In addition, other information such as undo/redo, text selection and location of the caret will be kept in the uiNamespace when you close the debug console, which means your editing experience is even more seamless during a single game session, while at the same time avoids cluttering the profileNamespace.

Currently, the save option, Control S (Ctrl+S), will simply save the contents of the debug console into profileNamespace. In addition, if you used the **Magic word** to edit a CT_EDIT control, the contents will be saved into that control as well (after preprocessing, to keep it compatible with Eden scripting fields, which don't support comments and preprocessor commands).

Open dialog

The open dialog allows you to open files from addons and functions.

The left-hand tree view shows the structure of the addon (parent-child), and the right-hand tree view shows the internal folder structure.

The tree view colors change from GREEN to RED to visualize the depth (level).

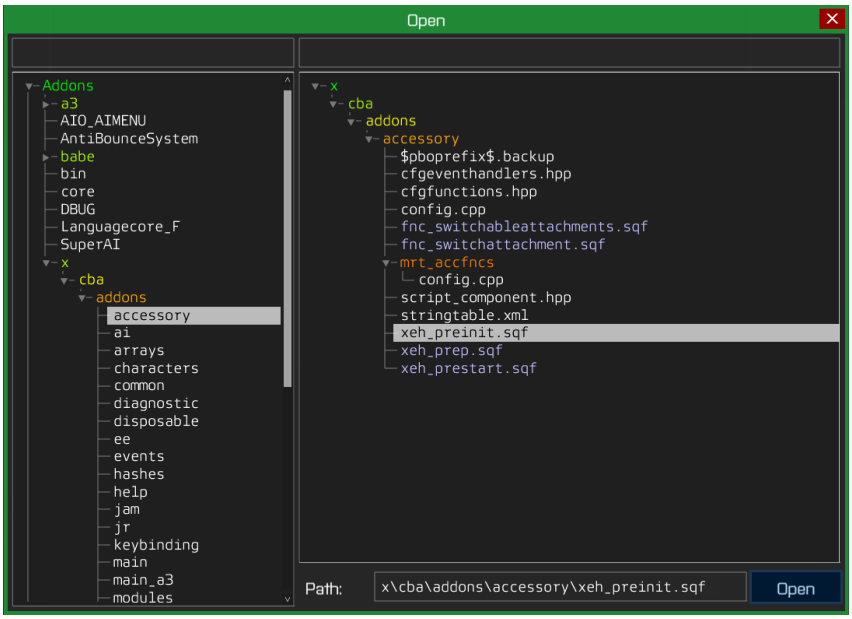


Figure 9 - The Open dialog

4.2. Code execution

Execution modes

There are 5 execution modes:

1. Local execution – unscheduled: 
   Executes the code using `isNil {ARGS call CODE}`.
2. Local execution – scheduled: 
   Executes the code using `ARGS spawn CODE`.
3. Remote execution – Global: 
   Executes the code using `[ARGS, CODE] remoteExec ["call", 0]`.
4. Remote execution – Server: 
   Executes the code using `[ARGS, CODE] remoteExec ["call", 2]`.
5. Remote execution – Custom: 

   Executes the code using `[ARGS, CODE] remoteExec ["call", TARGET]`. To define the target, right-click on the icon and type an **expression** which evaluates to valid target (see the remoteExec documentation on the Wiki). For example, you can type something like: `[0,-2] select isDedicated`, which evaluates to a number, which is a valid target for remote execution.

Arguments

It is possible to provide arguments to your code by typing an **expression** in the Arguments edit box. See the section on Execution modes for more details on how these arguments are passed to your code.

Figure 10 – Providing arguments to the code

Code performance

You can measure the performance of your code using the button. This will execute your code for a set number of cycles, and returns the average execution time in milliseconds (ms).

By default, the number of cycles is 10'000, but you can provide a custom number of cycles if desired.

Execution history

Every time you execute a code, a "snapshot" of the code will be saved in the profileNamespace. You can see the execution history using the Control H (Ctrl+H) shortcut, or alternatively, open the window through the File menu.

The codes are sorted by execution date, and placed in a tree view for easier access.

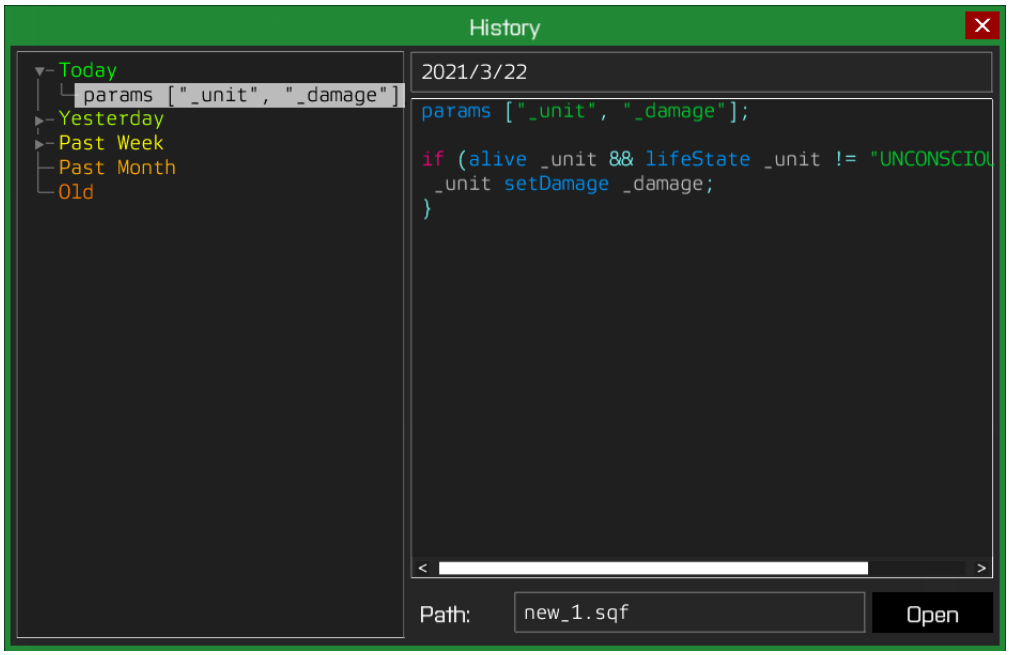


Figure 11 - Execution history

Preprocessor

The mod uses the SQF-VM preprocessor, which has a very good compatibility with the vanilla SQF preprocessor. However, note that some features (especially new ones) may not be available.

5. Watch/Draw

The watch/draw window is shown when you pause the game. The ⇆ button switches between watch/draw modes, and the ✚ button adds a new watch/draw.

Figure 12 - The top bar of the Watch/Draw tool. Notice ⇆ the and ✚ icons.

5.1. Watch

You can monitor a variable/expression using the Watch feature.

The Pin button pins the result of the expression to the left side of your screen, allowing you to track the results during gameplay.

You can also change the color of an expression for easier tracking.

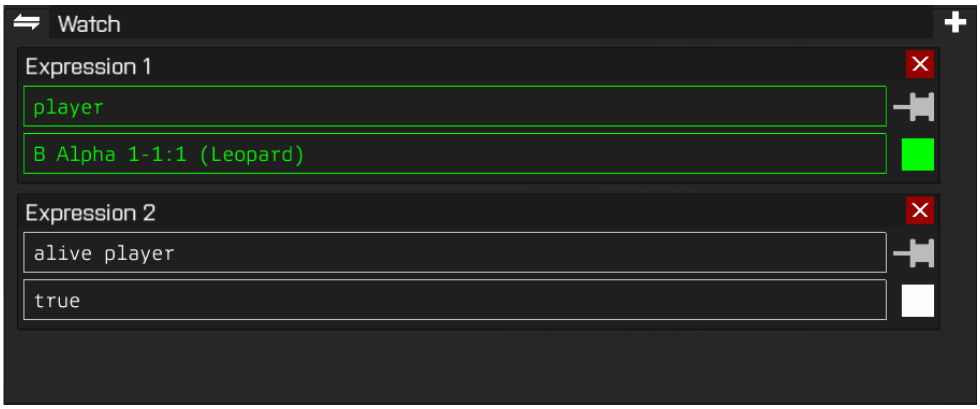


Figure 13 - Watched expressions and their results

5.2. Draw

This feature allows you to set up custom drawLines and drawIcons very quickly. It is useful when you'd like to track the results of something visually (such as positions, lines, paths, etc.).

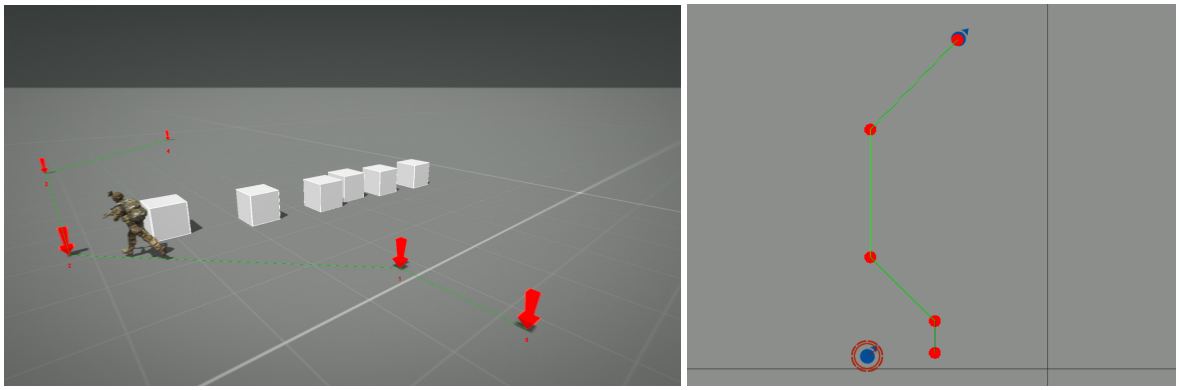
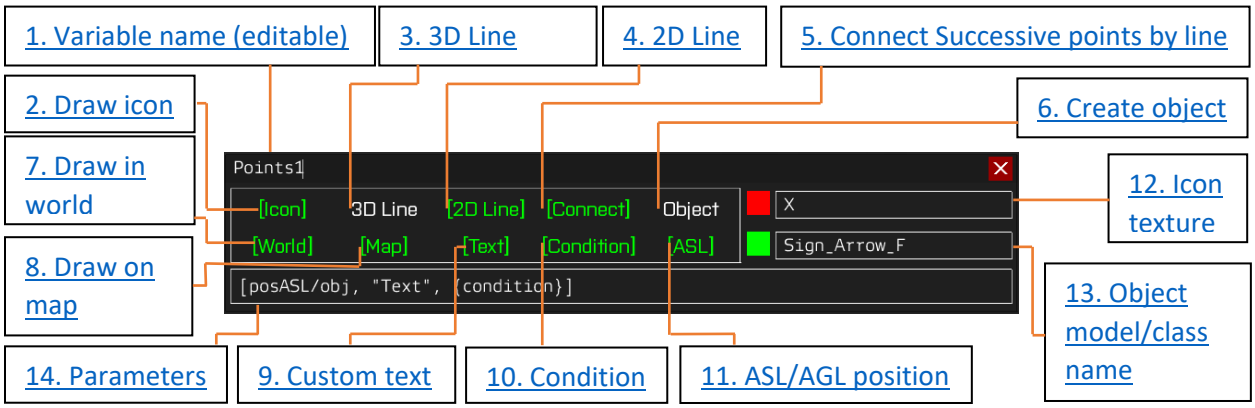


Figure 14 - Using the draw feature to show the AI path

In a nutshell, this feature creates a code for drawing icons, lines, and even objects in the world/map. The code is precompiled for faster performance. The user can add draw elements simply by adding elements

(such as positions and objects) to an array (e.g. using pushback). I will show an example at the end of this section.

The elements of each Draw window are as follows:



1. Variable name

This is the variable name used for the draw array. For example, in the above picture the variable name is `Points1`. `Points1` is an array, which you can fill with what you'd like to draw (as long as they are in agreement with the parameters field).

Every time you add a new draw window, the mod creates the variable names in format: `VariablePrefix#n`. The default variable prefix is **Points**, so the variable names will become **Points1, Points2**, etc. You can change the variable prefix in settings.

The variable name is modifiable and you can change it to whatever you prefer (e.g. `myAwesomePoints`).

2. Draw icon

This check mark adds drawIcon (for map) and drawIcon3D (for world) elements to the code.

3. 3D Line

When either 3D Line or 2D Line options are turned on, you can show line elements in the world or map. Note that despite the term 3D, it works both for map and world draw.

The term 3D simply refers to whether you want 3D line mode or 2D line mode in World draw (see below). So you can either check this or 2D Line, not both.

4. 2D Line

This sets the line mode to 2D. Because the command drawLine3D has some limitations, such as not visible through objects and showing very thin lines, you may want to use 2D lines instead.

Arma doesn't have a "drawLine2D" command. So this is a custom feature added by this mod.

5. Connect successive points by line

Normally, when you have the 3D Line/2D Line option selected, each element of the draw array must have a pair of points (**each element** must be in format `[startPosition, endPosition]`).

When this option is enabled (and 3D Line/2D Line must be enabled too), it automatically connects successive points using lines and you don't have to provide the start and end in each element.

For example, if the draw array is like: `[pos1, pos2, pos3, …]`, this feature connects pos1 to pos2, and pos2 to pos3, and …, using lines. Without this feature, the draw array would have to be like: `[[pos1, pos2], [pos2, pos3], …]`.

This feature is useful for showing **paths**, **polygons**, etc.

6. Create object

If you'd like to show some object at the position of a point, you can use this feature.

For example, this allows you to use the game's arrow sign object (which looks like ⇩) to show path points.

7. Draw in world

When this option is turned off, all draw elements that are shown in the world (Icon, 3D/2D Line and object) won't be created.

8. Draw on map

When this option is turned on, what you want to show (lines and icons) will also appear on the map.

9. Custom text

By default, the mod automatically adds the _forEachIndex (element index) as a text next to each point. If you'd like to show a custom text instead, you can use this feature (an empty string will remove the text shown next to a point)

Note that this requires **Icon** to be enabled.

10. Condition

If you'd like the icon/line to appear only when a certain condition is met, you can enable this feature.

Note that since the condition has to be dynamically evaluated, you have to supply a **code** that evaluates to a **boolean**.

Special variables _p1 (first point) and _p2 (second point, in the case of a line) are available for use.

For example, this will only show the point if the player is closer than 5 meters to it (assuming ASL format):

```
{getPosASL player distance _p1 < 5}
```

11. ASL/AGL position

By default, the mod uses the AGL position for drawing elements (since the commands drawLine3D and drawIcon3D require AGL positions).

However, since the AGL position is a relative position (height is relative to the ground), you may want to use an absolute position in your codes. The best absolute position format is ASL. So I've provided the option to directly use ASL positions without any need for conversions.

12. Icon texture

By default, each new draw window that you create uses the default icon defined in settings. This allows you to change the texture icon (when Icon is turned on).

In addition to supporting absolute paths to the texture (such as `\a3\...\somePicture.paa`), the mod also provides "magic names" that you can use instead of full texture paths. They are not case sensitive.

Magic names	Icon
Cross	
Circle	
Cursor	
Target	

If you'd like to scale the icon, you can insert a multiplication before the icon name.

For example: `2*target` uses the Target icon and scales it by a factor of 2 (spaces before and after the * sign are optional).

Warning! This is not a mathematical expression! Don't put () or any other signs (not even **/**) in the texture name!

13. Object model/class name

By default, each new draw window that you create uses the default icon defined in settings. If you'd like to show a custom object type, you can use the class name or path to model in here.

The mod also provides "magic names" for two common objects:

Sign_Arrow_F (recolorable): Use any of these: `arrow`, `a`, `v`, `>`, `<`

Sign_Sphere25cm_F (recolorable): Use any of these: `sphere`, `ball`, `b`, `o`, `0`. (the last two are the letter O and the number zero)

Similar to icon textures, you can also **scale** the objects if you want. For example, this will create the arrow sign half size:

```
0.5*arrow
```

Warning! This is not a mathematical expression! Don't put () or any other signs (not even **/**) in the model name!

14. Parameters

Once you've set up your draw, you can see the required parameters for **each element** of the array.

Example 1:

If the parameters box says: `posASL/obj`, all you have to do is add positions or objects to the array:

```
Points1 pushBack player
```

This will show the icon where the player is (updated automatically because you provided an object).

Example 2:

If the parameters box says: `[ posAGL/obj, posAGL/obj, "Text", {Condition} ]`:

```
Points1 pushBack [ [1500,3500,0] , [1640,3400,0] , "This is a line" , {player distance _p1 < 5 || player distance _p2 < 5}]
```

The above element will draw a line between positions `_p1=[1500,3500,0]` and `_p2=[1640,3400,0]`, shows the custom text `This is a line`, and will only show the line if the player is closer than 5m to either _p1 or _p2.

Example

Let's say I want to visualize the AI path points.

1. I want the array name to be AI_Path.
2. I don't want any icons, instead I want to show them using the arrow object. I want the points to be red.
3. Since I want the path index to appear, I won't turn off Icon, instead I clear the icon texture edit box.
4. I want the path points to be connected to each other using green lines.
5. I want 2D lines because I want to see the lines through objects.
6. I want to see the path both in the game and on the map.
7. Since the PathCalculated event handler returns the path in ASL format, I need to check ASL.

Based on the above requirements, this is the set up:

The only thing I need for every element is the position. So I add the "PathCalculated" event handler to the AI like this:

```
AI addEventHandler ["PathCalculated", {
    params ["", "_Path"];
    AI_path = _Path;
}]
```

And that's it! Now I can see the AI path every time I give him a move order!

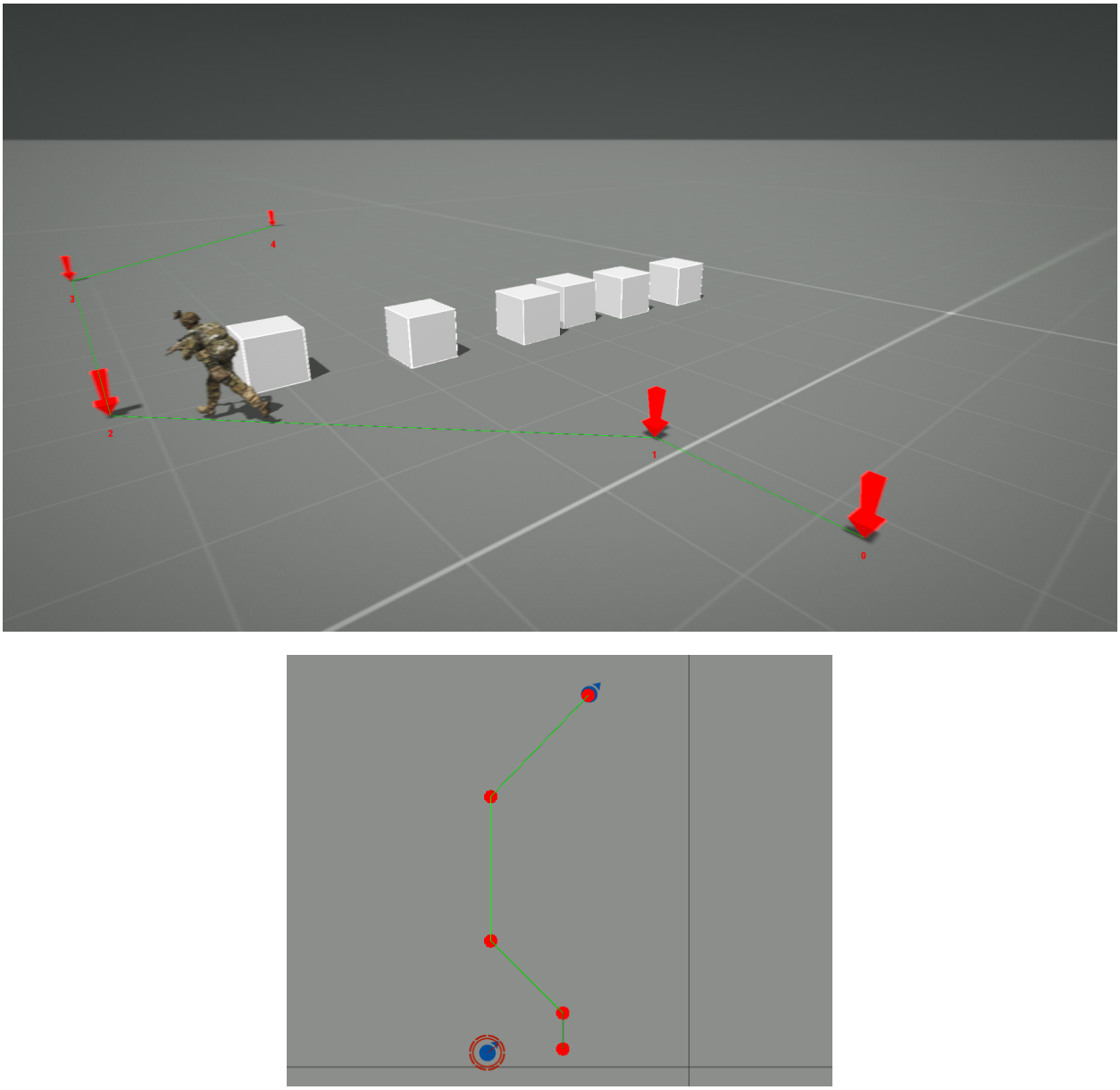


Figure 15 – Following the above example, AI path can now be seen in the world and on the map

6. Config viewer

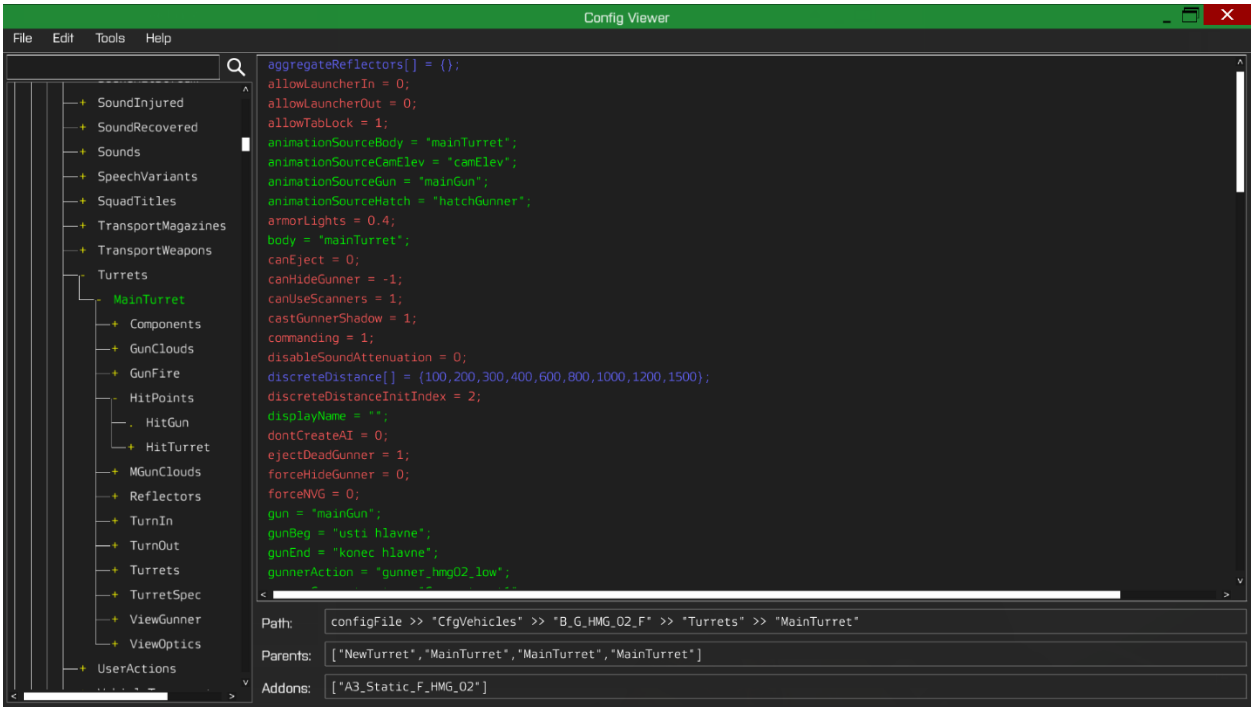


Figure 16 - The config viewer window

6.1. Navigation

Performance

You can seamlessly navigate through the config viewer without any annoyingly long loading times. The mod only “scans” the config entries when needed, and will cache the entries for faster browsing (it gets even faster the more you use it!).

Tree view

The config viewer features a custom tree view, which on the surface looks similar to the vanilla Tree View control.

The vanilla tree view has some major problems:

1. If the tree has a lot of entries, it slows down the further you scroll down.
2. When you perform a search, the tree levels and what’s visible is modified.
3. When you collapse a branch, all sub-entries are also collapsed.

My custom tree view doesn’t have any of those problems. I’m using something which I personally call **“smart rendering”**, which means it only creates controls in the visible part of the tree, which means **even if you have millions of config entries expanded, it won’t affect the scrolling performance**!

Back and forward

If you jump to a certain config entry by mistake, you can go back using Control Z (Ctrl+Z), and **go foward** using Control Y (Ctrl+Y) or Control Shift Z (Ctrl+Shift+Z).

Config path, parents and source addons

In addition to path and parents, which are also shown in the vanilla config viewer, you can also see the config addon sources, which shows which mods are modifying the config entry.

6.2. Bookmarks

The mod has a dedicated bookmarks window. The shortcut is Control B (Ctrl+B).

In addition to user bookmarks, the bookmarks window automatically adds several entries under the Default branch. They are detected based on the player's current vehicle, weapons, magazines, and ammo.

To bookmark the currently open config path, use the Control S (Ctrl+S) or Control D (Ctrl+D) shortcuts.

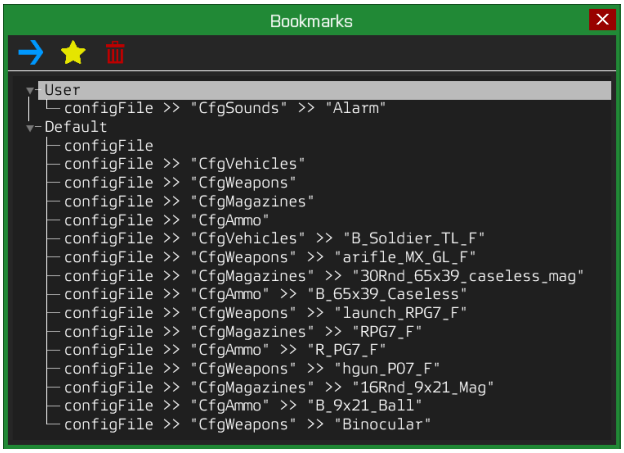


Figure 17 - Bookmarks window

6.3 Config search

Quick Search

The quick search feature is designed to give you very fast (but limited) search results, and it only seaches in class names. It has two search modes (both done automatically):

1. Config lookup (Exact match): It will search in common config classes (such as "`CfgVehicles`", "`CfgAmmo`", "`CfgMagazines`", "`CfgMovesMaleStr`", etc.) to give you an exact match. For example, if you search for AmovPercMstpSrasWrflDnon, it will direct you to `ConfigFile >> "CfgMovesMaleStr" >> "states" >> "AmovPercMstpSrasWrflDnon"`.
2. First match: In this mode, you can perform partial search (e.g. `CfgVeh` to go to `CfgVehicles`), but only in classes that are already scanned by the config viewer.

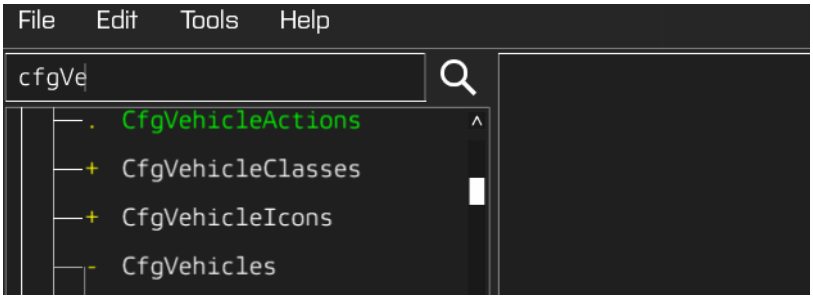


Figure 18 - The quick search text box and button

Deep search

The deep search mode allows you to search the config more thoroughly. It searches level by level and config to config, so it is very slow. It also supports going backward to the previous found entry.

The deep search is **"semi-scheduled"**. It means that it performs several searches together (**unscheduled**), but the search is being performed in the **scheduled** environment. This is done for two reasons:

1. Even though an unscheduled script can finish much faster that a scheduled script, since the deep search can take a significantly long time, you must be able to cancel the search midway.
2. The search can't be fully scheduled, because then it would take forever to finish.

You can specify the number of unscheduled searches in settings (default is 50). The larger this number, the faster the search can finish, but also the game FPS will also become lower, making it harder to stop the search.

You can open the Search window using the Control F (Ctrl+F) shortcut, or access it from the Edit menu.

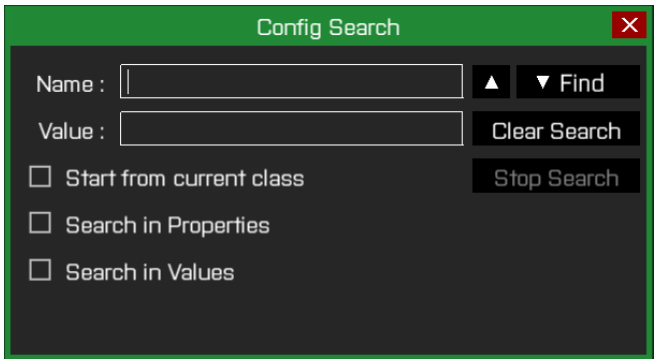


Figure 19 - Deep search window

Name: What to search in class name or property name

Value: What to search in property values (**Search in Values** must be checked)

Start from current class: It starts from the currently selected config entry and starts to go deeper from there. If unchecked, it starts from the root of the config (can be extremely slow!).

Search in properties: If checked, it also searches the property names for the text in **Name**. If unchecked, only class names will be searched (slightly faster).

Search in values: If checked, it searches in the config property values for the text in **Value**.

Clear search: The mod caches the last search results (e.g. for next and previous buttons). If you want to start a new search, it is recommended to always clear the last search results (if search parameters haven't changed, otherwise the mod will clear the last results automatically).

Stop search: Thanks to the "semi-scheduled" search algorithm, if you want to stop the search for whatever reason, you can use this button.

6.4. Data viewer

The data viewer allows you to see the contents of a config entry in its “processed” form. It opens automatically when you double click a config property.

Multiple data viewer windows can be opened at the same time.

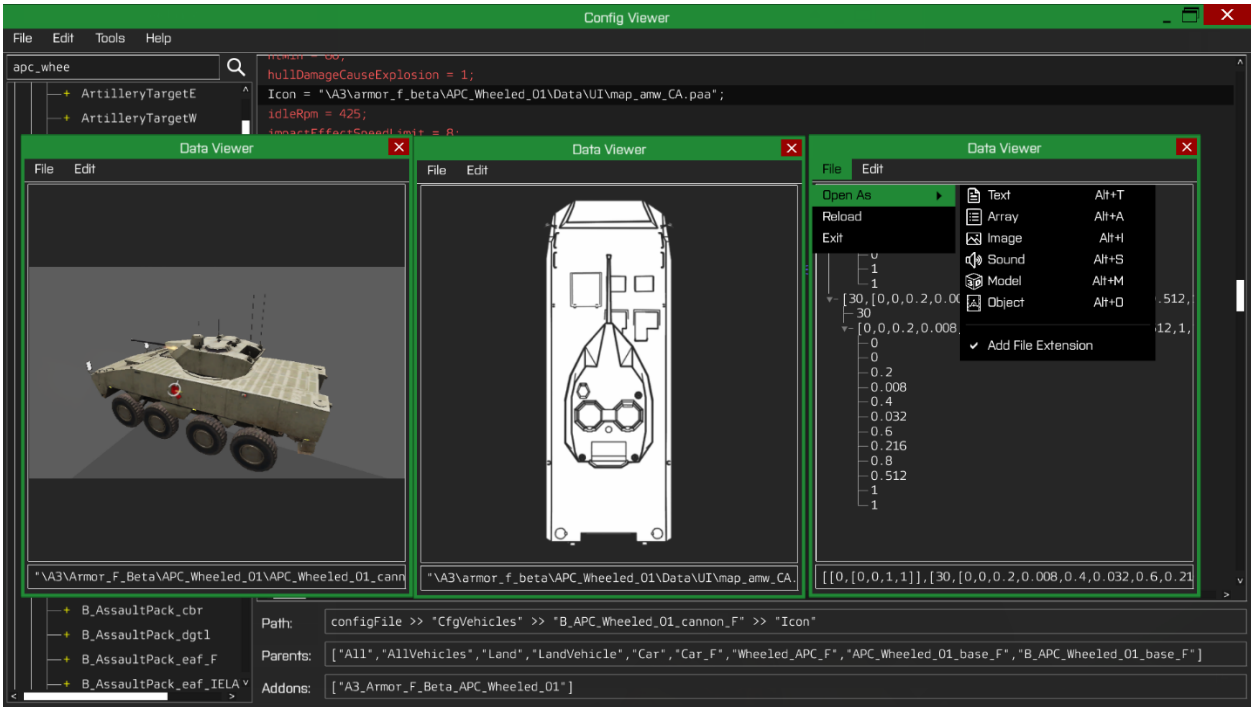


Figure 20 - Opening 3 data viewer windows at the same time.

It supports:

1. Plain text
2. Arrays
3. Pictures (.paa and .jpg)
4. Colors (array, procedural texture, and hex)
5. Models (.p3d)
6. Sound files (not working as intended)

If the mod fails to detect the appropriate data type or you want to open the data in a different format, you can use the Open As feature.

Open as Text

Plain data shown as a text. Useful for copying the raw data.

Open as Array

If the config data is in array format, you can open it as array, which shows the elements using Tree View.

You can double click any element to open them in a new Data viewer window.

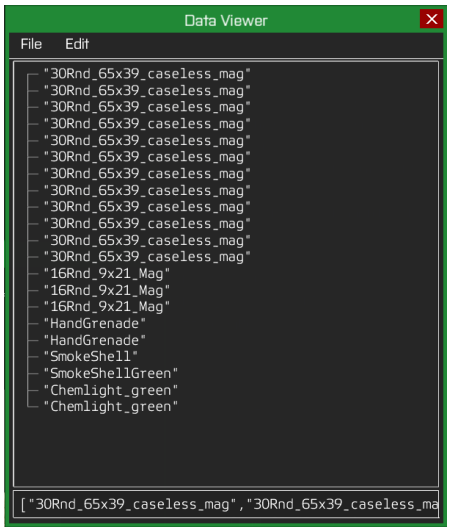


Figure 21 - The magazines of a unit, shown in array format

Open as Image

This feature can open:

1. Pictures (.paa and .jpg formats)
2. Color arrays (RGBA) such as [1,1,1,1]
3. Hex colors, such as “#FFFFFF”
4. Procedural textures, such as #(argb,8,8,3)color(1,1,1,1)”

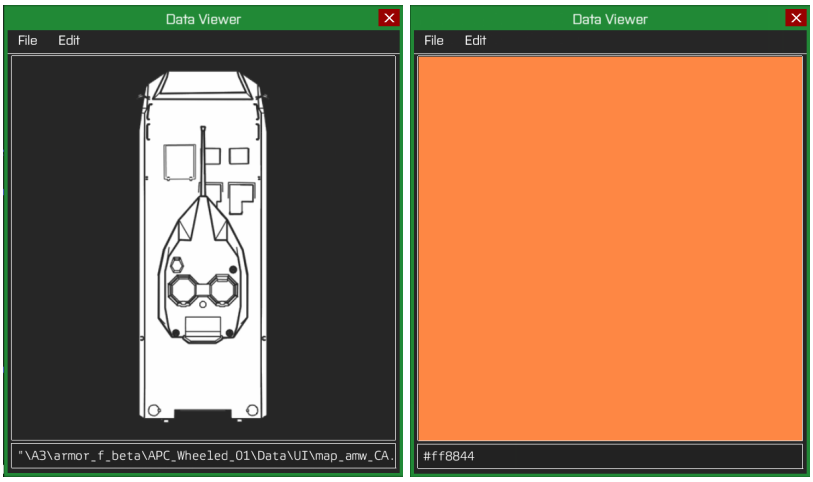


Figure 22 - Open as Image can handle both pictures and colors.

Open as Sound

Not working as intended, because Arma doesn’t support playing sound files directly.

It simply uses playSound3D command, which requires that the game be unpaused. The sound can’t be stopped.

Open as Model

Supports both cfgVehicle classnames and direct model paths (.p3d). Internally, it uses createSimpleObject.

You can move the camera by holding mouse left click on the window and dragging. You can also zoom in/out using the mouse wheel.

Figure 23 - The 3D model of a Marshal APC, in data viewer.

Open as Object

This is almost the same as Open as Model, but the difference is that it uses the `createVehicle` command to create the object (some objects don't have a proper simple object model).

Opening raw data

At the bottom of each data viewer window, you can see the raw data used to show the result.

You can change this data to anything you want, and view the data using the Open As feature.

Find in config

Under the Edit menu you can find the option Find in config.

It uses the config lookup feature (same as Quick Search) to find a matching config entry.

For example, you can start by opening a **CfgWeapons** sub-class. If you open one of the magazines in data viewer and use Find in Config, you go to the **CfgMagazines** entry for that magazine. If you open the ammo property and use Find in Config, you can go to the **CfgAmmo** entry for that ammo.

6.5. Go to config

The Go to config window can be accessed from the Edit menu or using the Control G (Ctrl+G) shortcut.

You don't necessarily have to enter the full config path, but only an expression that evaluates to a config (e.g. you can use `configOf player` to jump to the player unit's config).

7. Function viewer

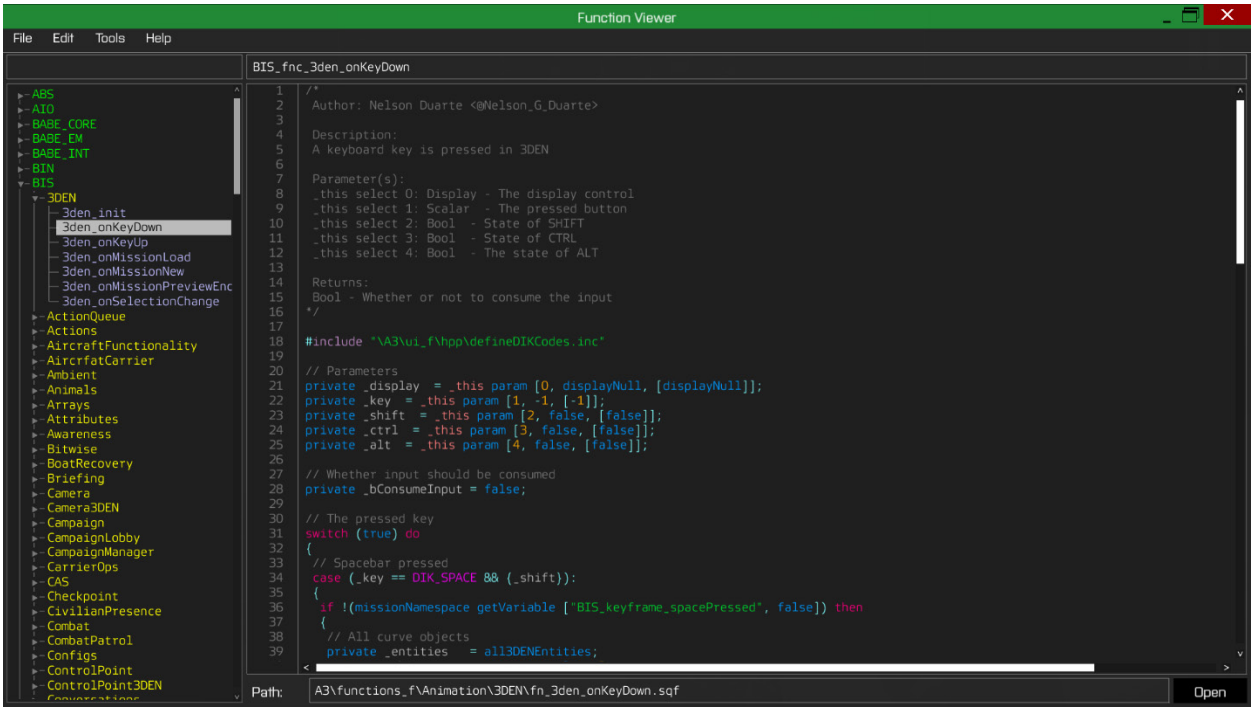


Figure 24 - Function viewer window

7.1. Tree view

All functions are properly categorized in a multi-level tree view.

The tree levels are Tags, Categories, and finally the function name.

7.2. Search in functions

You can find a function very quickly simply by typing part of its name in the search box.

Note that you should only put the function name in the search box (`TAG_fnc_` part should be omitted).

7.3. Syntax highlighting and line numbering

Read the contents of the function more properly using syntax highlighting.

Line numbers can help you find a line more quickly in the case of an issue (e.g. a function throws an error at a specific line).

Note that the function viewer doesn't have any parameter hints or command info. If you want these features, you can open the function in Debug console using the Open button (**don't copy-paste**!).

7.4. Search in function contents

The search feature is almost identical to the Debug console Search and replace, but there's no replace feature. You can open the search window using the Control F (Ctrl+F) shortcut.

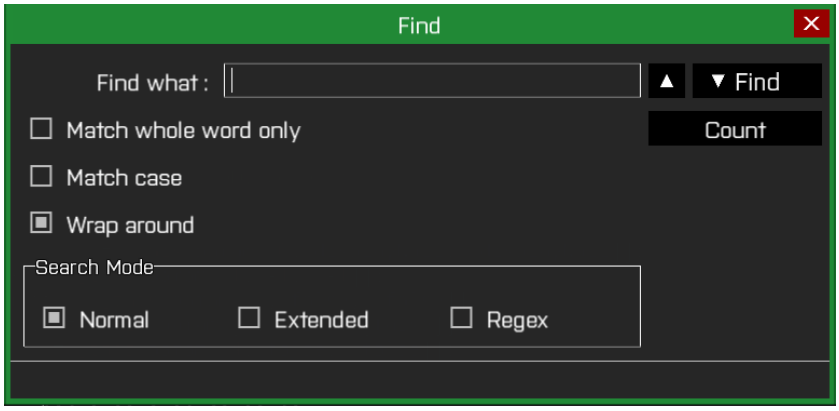


Figure 25 - The Find window of Function viewer

7.5. Function recompilation

Three recompile modes are available:

Recompile current

Recompiles the currently open function.

Recompile all

All functions will be recompiled.

Recompile sub-classes

Recompiles all functions starting from the current selection in the tree view.

This feature is useful for recompiling functions in a certain category or mod.

For example, the following tree selection will recompile the functions starting from `addWaypoint` to `taskSearchArea`.

Figure 26 - The tree view of the function viewer. Notice the selected tree item. Now all sub-classes (functions) can be recompiled.

If CBA was selected, all CBA functions (in all categories) would be recompiled.

8. Color picker

The color picker tool can be used for recoloring the parts of the mod that support theming. It can also be opened directly from the debug console using the Alt O (Alt+O) shortcut.

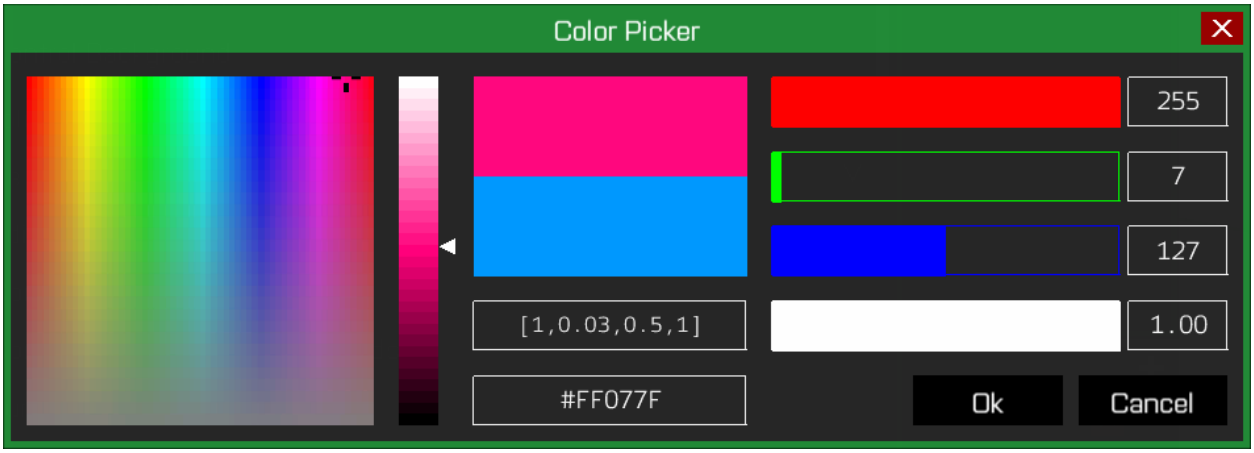


Figure 27 - The color picker window

8.1. Color palette

You can select colors more quickly from the color palette. The mouse cursor cannot be moved outside the color palette window (this is a design feature).

8.2. Color luminosity

You can change the color luminosity using the vertical luminosity bar.

8.3. Individual color channel sliders and edit boxes

You can change the RGBA color channels separately using the provided sliders/edit boxes. The color will update automatically.

8.4. Undo color changes

The original color information is kept, and is shown below the currently selected color.

To undo the color changes, simply click on the original color.

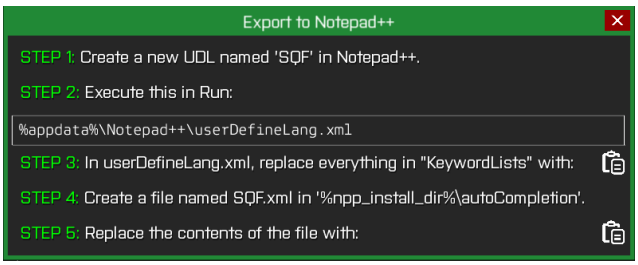
8.5. Hex and RGBA color formats

The hex and RGBA (percent) color formats are the most frequently used in SQF scripting.

As you change the color, these color formats are also updated and shown in the text boxes below the color.

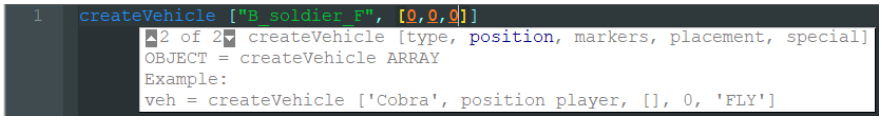
You can also insert the colors in these formats into these fields, and the color picker will automatically read the color and set the slider values.

9. Export to Notepad++



This feature allows you to create (or update) a SQF user-defined language (UDL) in Notepad++. Currently it has to be done manually, but in the future I might make it automatic.

In addition to auto-completion and syntax highlighting, it also exports the command parameter hints, but it only works for commands with a right-hand array parameter, and it only shows after typing square brackets [ ].



It will also export your currently selected functions.

Note that it **doesn't export the colors**, so you'll have to change them manually later (needs to be done only the first time you do this).

Step 1: Create a new UDL named "SQF" in Notepad++

You can skip this step if you already have one.

If you create a new one, please close Notepad++ after that.

Step 2: Open the userDefineLang.xml file

You can simply execute this in Run (⌘ R Windows+R) and it'll take you there:

`%appdata%\Notepad++\userDefineLang.xml`

Be sure to **make a backup** of this file first (File > Save a Copy As).

Step 3: Open the Export to Notepad++ window

You can find it here:

`Debug console > Tools > Export to Notepad++`

Step 4: Replace contents in the UDL (read thoroughly!)

Copy the exported keywords in the game (click on the clipboard button) and go back to the userDefinedLang.xml file.

Find the SQF UDL (`<UserLang name="SQF"`), select everything **under `<KeywordLists>`** (see the picture below), and then paste.

Note that all keywords under KeywordLists must be present after this step. If you notice a missing keyword, undo the change (note that order of keywords does not matter).

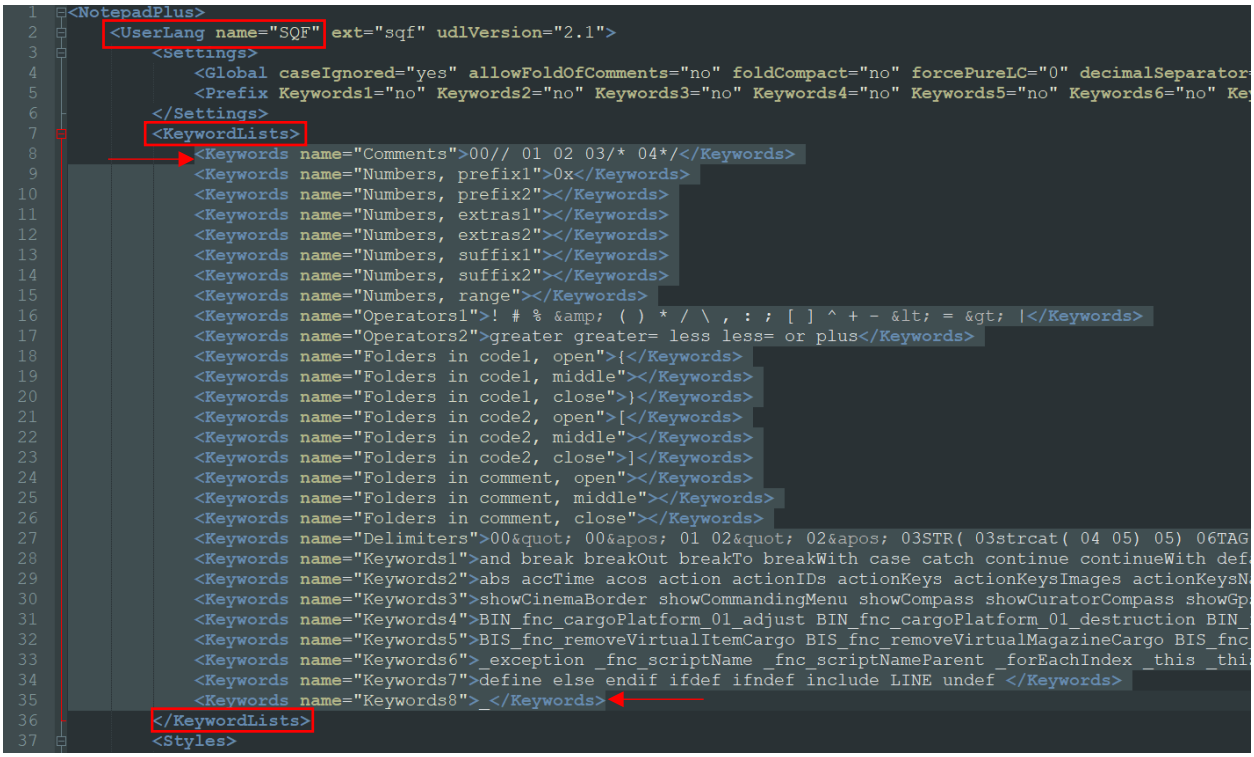


Figure 28 - userDefineLang.xml - Notice the UDL name, and the selection start and end positions, which select everything after the <KeywordLists> and before the </KeywordLists>

Step 5: Create the auto-completion file

Go back to the game and copy the autocompletion data (the second clipboard button).

Navigate to your Notepad++ installation folder (default is: **Program Files (x86)\Notepad++** for 32 bit version and **Program Files\Notepad++** for the 64-bit version).

In the installation folder you should see the **autoCompletion** folder.

Create a new file named **SQF.xml** (if it doesn't already exist), and delete the entire contents of the file and paste the new contents from the clipboard.

10. Settings

The mod features an extensive list of settings to customize the mod. If I thought some parts of my design are too subjective, I added the appropriate settings for the users to change them to their liking.

10.1. Debug console settings

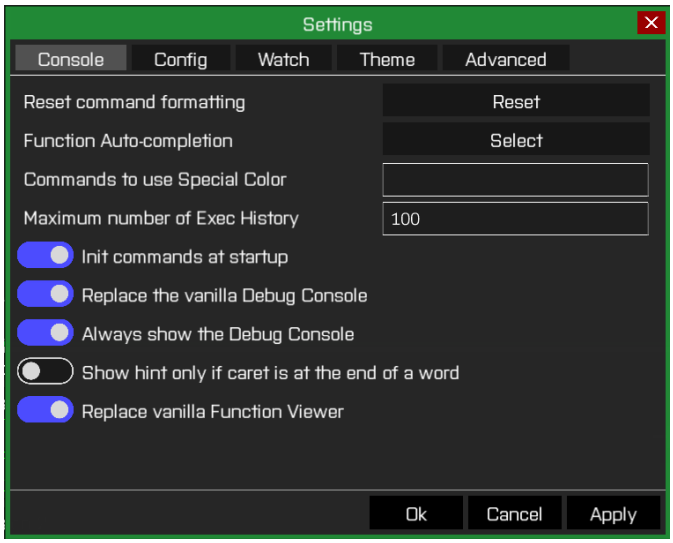


Figure 29 - Debug console settings

Reset command formatting

Re-reads the commands and applies their formatting. It's for debugging purposes.

Function auto-completion

By default, only BIS and BIN functions are available for auto-completion. If you want to add more functions such as CBA and ACE for auto-completion, you can add them in here.

Commands to use Special color

If you want some commands to be in the Special color (which is used only for control structures, such as if then), you can add them in here.

For example, if you want to add true and false, put: `true;false` in the edit box (they can be separated by space, comma, or semicolon).

Maximum number of exec history

Sets the number of execution "snapshots" to keep. Note that larger numbers will clutter your profileNamespace.

Init commands at startup

Detects the commands at game startup.

Replace the vanilla Debug console

The mod's debug console will be used instead of the vanilla debug console.

Always show the Debug console

The mod's debug console will appear in every mission when you pause the game, even if the debug console for that mission is disabled.

Show hint only if caret is at the end of a word

By default, the mod shows the parameter hints when the caret is anywhere on a word. You can make it only appear when the caret is placed at the end of the word.

Replace vanilla function viewer

Use the mod's function viewer in the Tools menu.

10.2. Config viewer settings

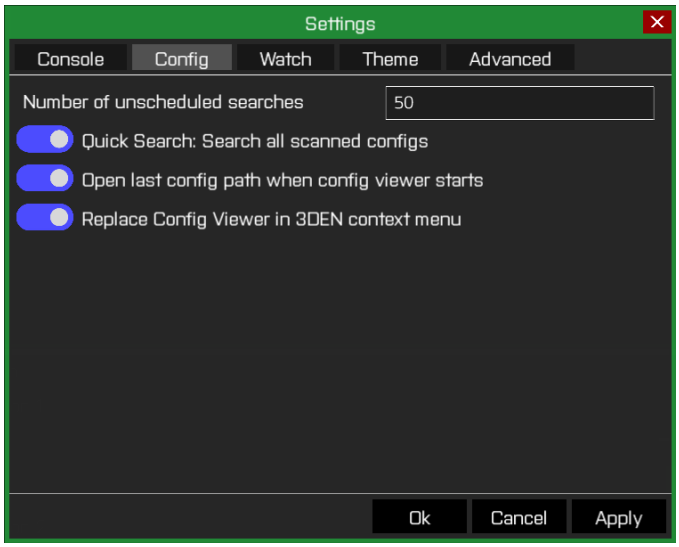


Figure 30 - Config viewer settings

Number of unscheduled searches

Sets the number of unscheduled searches. Higher numbers will speed up the search but cause longer freezes.

Quick search: search all scanned configs

If disabled, quick search will only search in the configs that are in the expanded tree view. It is faster but obviously if something is not in view you can't find it anymore.

It is recommended to keep this option on.

Open last config path when config viewer starts

Opens the config viewer to the last viewed config.

Replace config viewer in 3DEN context menu

When you right click on an entity in 3den and use the Find in Config option, it uses the mod's config viewer which is significantly faster.

10.3. Watch settings

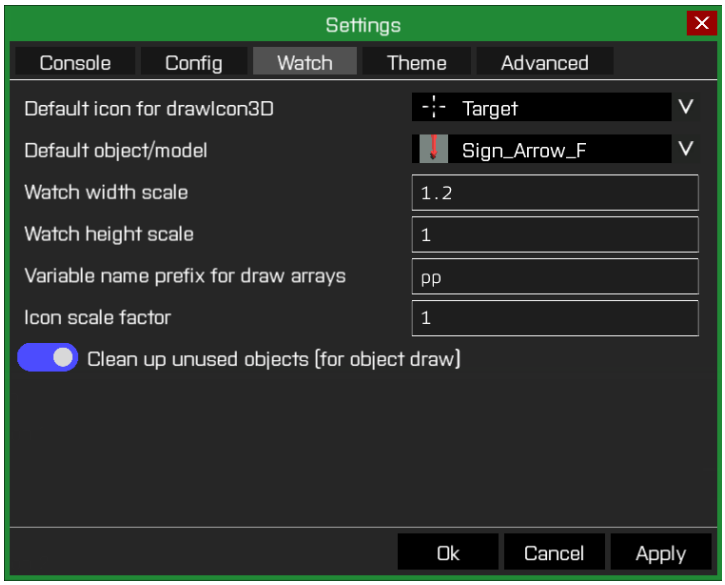


Figure 31 – Watch/Draw settings

Default icon for drawIcon3D

The default icon to use with Icon draw mode.

Default object/model

Default object/model path to use for creating objects.

Watch width/height scale

The watch window will be scaled by this factor. The debug console will also move/resize accordingly.

Variable name prefix for draw arrays

Sets the variable name prefix for draw arrays. It must follow the variable naming convention (e.g. no spaces, not starting with numbers, etc.). Also note that since this is a global variable, you can't start it with underscore either.

Icon scale factor

Sets the default icon scale. For individual icon scale factors, see the section on icon textures.

10.4. Theme settings

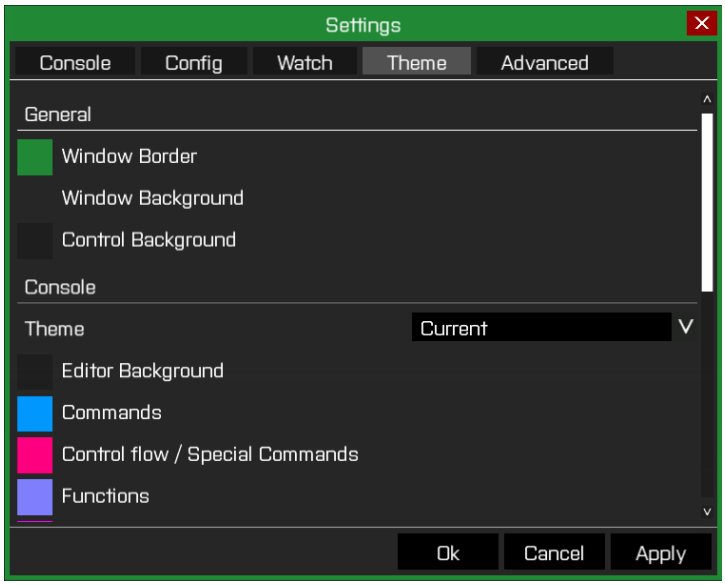


Figure 32 - Theme settings

You can change the colors of various elements here.

10.5. Advanced settings

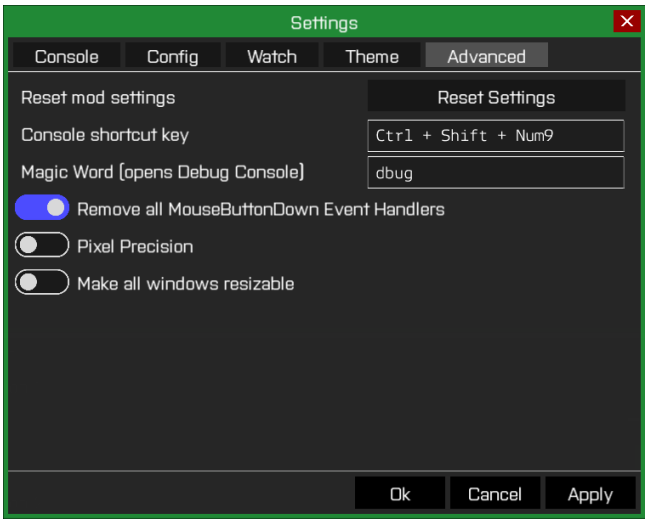


Figure 33 - Advanced settings.

Reset mod settings

Delete all mod settings stored in missionNamespace/uiNamespace/profileNamespace.

Console shortcut key

If you want to open the Debug console using a shortcut, you can set it here.

Magic word

Sets the magic word. Magic word is not case sensitive.

Remove all mouseButtonDown event handlers

It is needed for the debug console to work properly when it replaces the vanilla debug console.

Pixel precision

Pixel precision is very similar to anti-aliasing. It smooths text edges but may make them blurry.

Make all windows resizable

All windows will become resizable, but many will be buggy.

11. Known issues

Window priority

The order of window will depend on the order in which they were created. This can cause some issues when you have more than two windows open and you switch to a previous window.

For example, if you open Win1, Win2 and Win3 in that order, and switch to window Win1, and then switch to Win2, Win3 will still be on top of Win1.

Note that this won't create any issues with the window you're currently working with. So it's a minor issue.

This is an Arma feature and I can't fix it.

Occasional crashes when you create new displays from the debug console

Sometimes when you execute a code that creates a new display, the game might crash. To circumvent this issue, execute the code scheduled.

The vertical and horizontal scrolls don't update when you resize the windows

Not all that important but it can be a bit annoying. When the contents of the window change, the scrolls will update as well.

12. Frequently Asked Questions (FAQ)

Q1. I disabled all options and now I can't open the debug console anymore. What should I do?

A1. Use the magic word feature. Type the magic word into any text box to make the debug console appear again.

Q2. What language did you use to write this mod?

A2. All parts except for the preprocessor and regex are written in SQF. The preprocessor and regex were written in C++.

13. Credits and thanks

Thanks for reading through the document. This should've helped you use the mod more effectively.

You may also have noticed the extensive level of details and attention that was put into this work!

If you like my work and would like to support me, please check out my Patreon page.