

# **SQDev Manual**

Raven

Last Modified: January 2, 2018

# Contents

<b>Preamble</b>	<b>iv</b>
<b>1 Installation</b>	<b>1</b>
1.1 Download + Prerequisites . . . . .	1
1.1.1 Prerequisites - Eclipse . . . . .	1
1.1.2 SQDev . . . . .	2
1.2 Installation . . . . .	2
<b>2 Getting Started</b>	<b>4</b>
2.1 First start . . . . .	4
2.2 New Project from scratch . . . . .	4
2.3 New project from existing mission/project . . . . .	5
2.4 Workflow within a SQDev project . . . . .	5
<b>3 Introduction to Eclipse</b>	<b>7</b>
3.1 Basics . . . . .	7
3.1.1 Terminology . . . . .	7
3.1.2 UI structure . . . . .	8
3.1.3 (Re-)Opening a specific view/perspective . . . . .	9
3.2 Customizations . . . . .	10
<b>4 Feature Overview</b>	<b>11</b>
4.1 Implemented Features . . . . .	11
4.1.1 SQF-Editor . . . . .	11
4.1.2 StringTableEditor . . . . .	14
4.1.3 RPT-Viewer . . . . .	14
4.1.4 Wizards . . . . .	15
4.1.5 Config- and Header-Editor . . . . .	19
4.1.6 SQDev-File-Editor . . . . .	19
4.1.7 SQDev-Perspective . . . . .	19
4.1.8 Linking . . . . .	19
4.1.9 Command-update . . . . .	20
4.2 Future Plans . . . . .	20

## Contents

<b>5</b>	<b>Configuring the plugin</b>	<b>21</b>
5.1	Settings . . . . .	21
5.2	Configuration Files . . . . .	25
5.2.1	SQDev-files . . . . .	25
5.2.2	link.sqdev . . . . .	27
5.2.3	SQFKeywords.txt . . . . .	28
<b>6</b>	<b>Tips and Tricks</b>	<b>29</b>
<b>7</b>	<b>Contributing</b>	<b>30</b>
7.1	Reporting Bugs . . . . .	30
7.2	Feature requests + Improvement suggestions . . . . .	30
7.3	Maintain the BIKI . . . . .	31
7.4	Contact . . . . .	32
<b>8</b>	<b>Version History</b>	<b>33</b>

# Preamble

I started working on the SQDev plugin back in 2014 because I was just getting into the SQF scripting language and it bothered me that there was no real IDE for it.

I knew about IDEs because I played a little with Java in Eclipse before, so I knew that there were tools that will point out syntactic errors while writing the code. The lack of such tools got especially notable because in ArmA or rather in SQF you have to open the game, load your mission/mod just to get some sort of cryptic error message. Then you'd have to go back to your source code and find the error (more often than not it was simply a forgotten semicolon).

So as Eclipse was the only IDE I knew at that time I started looking for a plugin that would allow me to write SQF code in it. It didn't take long until I stumbled upon the ArmADev plugin which promised to deliver that exact functionality. Delightedly I installed the plugin, loaded a script file in it and deleted a semicolon. And then ... Nothing happened. There was no error marker in the editor although I just created an obvious error in the source code.

It turned out that ArmADev simply delivered the possibility to write SQF code in Eclipse but didn't implement any syntax or type checking.

So I started looking into Eclipse plugin creation and not long after that I found out about the Xtext-framework which is some sort of "plugin-generator" for Eclipse. It seemed easy to use so I started working with it for the next year and year and a half. In parallel I self-taught myself to program in Java by writing a program that would strip all SQF commands from the BIKI (I noticed really soon that retrieving that data by hand is not really doable). After I got that working I tried to write a program that would transfer the gathered information into a grammar out of which Xtext would build the actual plugin.

After about two years of work on the plugin I still couldn't get it to work and I finally found out that my approach to the problem wouldn't work out (I tried

## *Preamble*

to solve everything via the grammar or rather with the created parser, because I didn't misunderstood what a parser is for). Additionally all the code written 'till then was really messy and not really object-oriented due to the fact that this was the code I learned how to program. Therefore I decided to give up everything and restart from scratch. But this time I didn't want to be dependent on some third-party software so I decided to write the plugin by hand.

As this was something I haven't had any experience with at all I started slowly and rescheduled the overall goal of syntax and type checking for some time (turned out to be quite a lot of time) into the future.

So it all started with an editor that provided simple syntax highlighting. In addition to that there was a wizard for creating a new project or a SQF file. And that was it. That was the first version of SQDev. Since then I continuously added one (little) Feature at the time. I slowly made my way towards my first and original goal for this whole project: Syntax and Type checking during code development.

# 1 Installation

## 1.1 Download + Prerequisites

### 1.1.1 Prerequisites - Eclipse

First of all you need to download and install Eclipse on your system. Which version of eclipse you are installing doesn't really matter as SQDev doesn't have any dependencies on cutting edge eclipse-frameworks. So if you happen to have an eclipse installation on your system you can simply use that.

However if you will install a fresh version of Eclipse anyway you can just as well install the latest version. You can download it here:

- <http://www.eclipse.org/downloads/eclipse-packages/>

You can try using the installer but I tried it twice and it didn't work in both cases (but maybe that's a Linux problem). If you don't want to or can't use the installer you can select one of the packages (I'd suggest "Eclipse IDE for Java Developers") and download it. It will download a zip-file that already contains the "installed" Eclipse. That means it can run out of that folder (as soon as it's unzipped).

However if you want to "install" it on your system you have to move the folder into your `C:\Program Files` or `C:\Program Files (x86)` (for a 32-bit installation) directory (on Windows) or in `usr/share` (on Linux). You can then create a shortcut to the `eclipse.exe` (Windows) or the `eclipse` (Linux) executable and add it to your desktop (if desired).

If you are running Linux you should check your system-repository first as it is most likely present in there and it's quite easier to install it from there.

### Workspace location

When starting eclipse for the first time it will ask you for a location for the “workspace”. This is the location where Eclipse will store all files and projects created within eclipse on disk.

It might be tempting to set the workspace to the same location as the Arma (mission-) directory but this is not recommended. The plugin is designed for an external workspace with no exceptions. **Therefore you should create a new workspace outside of any Arma directories** (for example in your *Documents* directory) and use that one.

### 1.1.2 SQDev

The download of the SQDev plugin is fairly easy. You can get it directly from GitHub via this link:

- <https://github.com/Krzmbzrl/SQDev/releases>

Unless you have specific reasons not to you should always download the latest release of the plugin.

## 1.2 Installation

The installation of the plugin itself is pretty straight forward as well. The following steps will show you how to install SQDev.

1. Open **Help - > Install New Software...**
2. In the appearing pop-up select **Add** (right next to the **Work with:** text field)
3. This will open yet another pop-up in which you have to provide a **Name** for the installed plugin which in this case is "SQDev"
4. In the next step you have to select the **Archive...** button and then navigate to the zip-file of the plugin you have downloaded

## 1 Installation

5. Confirm with
6. Back in the first opened window you should now have the ability to select SQDev in the big middle window
7. Uncheck the ☐  as this will slow down the installation a lot
8. Click  twice and then accept the licence (If this is the first plugin you are installing on this Eclipse installation it might ask you to agree to the Eclipse licence first)
9. Hit  in order to start the installation
10. Eclipse will warn you about the fact that you are about to install an unsigned plugin. This is absolutely normal (because I haven't looked into signing yet) and can be bypassed by clicking
11. After the plugin has been installed you need to restart Eclipse (It should suggest that by itself anyway)
12. Congratulations you just have successfully installed SQDev



## 2 Getting Started

If you haven't worked with Eclipse yet I'd suggest you to start with chapter 3 on page 7 in order to get to know the basic terminology and a feeling for the UI structure.

### 2.1 First start

If started for the first time Eclipse will greet you with a welcome window which you can skip by clicking the `Workbench` button in the top right corner.

If you just have installed the plugin you need to go into the preference menu (`Window` » `Preferences`). All relevant preferences are on and underneath the `SQDev` node.

If you don't want to customize anything the absolute minimum is to pay the `SQDev` node a visit and make sure the plugin found the proper ArmA directories. You might also want to set the other preferences on this page as they are affecting the plugin behavior (not just some cosmetic properties). More on that subject can be found in section 5.1 on page 21.

### 2.2 New Project from scratch

In order to create a new project you can use the included wizard by right-clicking into the navigator and then selecting `New` » `SQDev Project` (if you are already in the SQDev perspective) or `New` » `Project...`, then scrolling down to the `SQDev` category and selecting the `SQDev Project` option.

After this the wizard should open a window which allows you to provide some further information about the project to be created. After having provided the respective information the project will be created with a click on **Finish**.

After the project has been created it will appear in the navigator. If you are not in the SQDev perspective the plugin will ask you whether you want to switch to it when creating the project. Unless you have a reason not to do so I'd suggest you to perform the switch.

### 2.3 New project from existing mission/project

If you already have a mission/project that you'd like to work on in SQDev you can do so by importing it into the eclipse workspace.

For that you have to open the context menu in the navigator and select **Import...**  
**Import as SQDevProject**.

In the appearing dialog you have to provide the path to your project and then import it by hitting **Finish**.

This will now copy the selected project into Eclipse's workspace and lets you work on that.

#### **SQDev perspective**

If importing a project the plugin will not offer to automatically switch to the SQDev perspective (if you're not already in it). If you want to do so you have to do it manually.

### 2.4 Workflow within a SQDev project

Once an SQDev project has been created it can be manipulated just like any other folder in the filesystem of your computer. You can access the respective functionality via the context menu of the respective project folder (or any folder inside it).

## 2 *Getting Started*

Under **New** you can add new files and folders. If you are about to add a new SQF file you can either create an empty one by yourself by using **New** **File** or you can use the respective wizard shipped with the plugin.

In order to do so you have to select **New** **SQF File**. The advantage of doing so is that the wizard will pre-generate a header template filled in with some information about it, the containing project, the author and some more information (Details on that can be found in section 4.1.4 on page 15).

If you are wondering how your changes are actually transferred to the ArmaA directories you can have a look at how the plugin handles Linking in section 4.1.8 on page 19.

# 3 Introduction to Eclipse

## 3.1 Basics

### 3.1.1 Terminology

- **View:** A view in eclipse is basically an extra window embedded in the main Eclipse window next to the editor itself. They can typically be found to the left, right and underneath the editor. If there is more than one view on that specific side they either divide the available space so that each view has some fraction of the available space or they stack up in a register. In that case the user can switch between them by clicking on the respective header of the view.

The fact that views are being described as “windows inside a window” is due to the fact that they provide all basic functionality of a stand-alone window. They can be maximized, minimized and closed. Maximizing and minimizing can either be accomplished by double-clicking the views header (the part that displays the view’s name) or by using the respective buttons that can be found to the right of each view’s header (or to the far right end of the stack in case of stacked views).

- **Perspective:** A perspective in Eclipse is a preset of views in combination with some customizations of the toolbar and menu.
- **Navigator:** The navigator in Eclipse is basically what is called a “file browser” in other software. It is a special view (typically to the left of the editor) that lets you see all existing projects existing in the current workspace. It is also possible to apply filters in order to sort out projects (or files within projects) that aren’t wanted to be seen.

Most Eclipse plugins ship with their own variant of a navigator (most time

they have special names, e.g. “package explorer” for the Java Development Tools) which allows an adaption of the context menus and available filters (also what filters are turned on by default).

The navigator will be the view you will be working with the most (except the editor itself) as this is the place where you can manage your projects and their content. All creation of new files is handled in there.

#### Custom SQDev navigator

Currently a custom navigator for SQDev is under development. It is however already included in the current version of the plugin as a sort of alpha-version (it has to be manually opened though).

### 3.1.2 UI structure

The basic UI structure isn’t actually that complicated. It mainly consists of six components:

1. **Menu:** The menu is the same as in almost every other software available out there. It is located at the very top of the window and provides access to several sub-menus.
2. **Toolbar:** The toolbar is located right underneath the menu and normally consists of buttons with various images on them.

Typically the content of the toolbar is perspective-dependent to some degree, meaning that some buttons will (dis-)appear when working in a specific perspective. Therefore the toolbar is the main location to look for when searching for utility-functions that are associated with the perspective you’re working in.

3. **Editor:** The Editor is the main part of eclipse and is located in the center of the window.

If several editors are opened simultaneously then they will be stacked in that centered part of Eclipse. It is even possible to open two editors next to each other by dragging them accordingly (see below).

### 3 Introduction to Eclipse

4. **Navigator:** The navigator is located (by default) to the left of the editor. If more than one navigator is opened at the same time, they get stacked there the same way editors are being stacked.
5. **Views:** Views are distributed around the editor and are typically stacked because there are too many of them to all be opened next to each other.
6. **Perspective list:** This part is actually fixed in the top right corner (right next to `Quick search`) and provides a button-list of all currently opened perspectives and therefore allows quick switching of different perspectives.

Via right-click these perspective-buttons provide further functionality. The most important of those are `Close` and `Reset` (which will reset the whole UI to the defaults of the given perspective).

As mentioned in the previous section all of these little “sub-windows” inside the Eclipse-UI provide most of the default window behavior one would expect from any top-level window.

That means they can be maximized by double-clicking on their header (where the `x`-button for closing it is located). Furthermore it is possible to drag them to a different location in the UI with that header.

If you drag them next to another header, the dragged one will be added to the respective stack. If being let go about in the middle of another sub-window (of the same kind) the available space is being split in order to show both windows in parallel. This can be reverted by re-stacking the respective window(s).

#### 3.1.3 (Re-)Opening a specific view/perspective

##### Opening Views

While working with Eclipse it will sooner or later happen that you accidentally close a view you actually needed. Or maybe you found out about a new view you want to open.

In order to do this you can go to the Menu `Window >> Show View`. The you are being presented with the default views associated with your current perspective.

### *3 Introduction to Eclipse*

If the desired view is contained in that list you can simply click on it and it will be opened for you. Otherwise you need to select **Other...**. In the now opening window is a list of all available views (you can use the search-bar in order to find the desired one).

#### **Perspectives**

A perspective can be opened almost the same as a view. The only difference is that you need to use **Window > Open perspective**.

A newly opened perspective will directly be activated. If you want to switch back to another perspective that is already open, you can do so by clicking by the associated button in the perspective list in the top right corner.

## **3.2 Customizations**

Coming soon

# 4 Feature Overview

## 4.1 Implemented Features

### 4.1.1 SQF-Editor

The heart of the whole plugin is the SQF-Editor. It allows the editing of SQF-files while providing a variety of further features that exceed a normal text editor and (most) other available SQF-specific editors.

As part of these features it does also support the use of macros inside your code. They will not interfere with the editor's features and are to a great extent even included in them.

#### **Syntax highlighting**

As any decent editor for a programming language the SQF-editor does provide syntax highlighting for the SQF source code.

It differentiates between Commands, Strings, local variables, global variables, macros and comments. The color for each of these types can be customized in the preferences of the plugin (see 5.1 on page 21).

#### **Content Assist + Auto-Completion**



The editor does provide content assist that can be invoked by `Ctrl+space`. It will bring up a pop-up listing all available inputs (starting with the letters already being typed in).

Those inputs can be macros, variables or commands. If available the plugin will show another window next to those proposals with additional information about the currently selected proposal (for commands that would be information about



## 4 Feature Overview


the command as can be found on the respective wiki-page). For commands these windows are the same as the hover-info covered in the next section.

The info- and proposal window can either be closed by triggering the insertion of the currently selected proposal via  (the focus needs to be on the proposal-window for that) or double-click, by clicking elsewhere or by hitting .

### Hover-Info

When hovering the mouse over a command a pop-up will be opened containing information about that specific command. It will contain the (beginning of the) command's description, one (of possibly more) syntax of this command, some locality information as well as its return value.

When hovering the mouse over this window or clicking on it it will switch to its “extended-mode” in which it will provide several tabs to select from each of which will cover another topic on this command. Essentially this window contains all information that are available on its respective wiki-page.

It does also provide a button  in the lower left corner that will actually take you directly to said wiki-page in case you want to pay it a visit.

### Syntax check


Here it starts to get interesting. While you are writing your SQF code (or at some specific trigger-events) the plugin will perform a syntax check on the written code.

This means that it will check whether the used commands are actually used in a way they will work. That means it will issue an error if a unary command (takes one argument to the right) has been used as a nular (no arguments) one.

In fact it will also issue an error for missing semicolons and the cool part is that this will not be in some cryptic way as ArmA tends to do.

All these errors are listed in the Problems view and (more importantly) marked directly in the editor by red underlining the respective part in the source code as well as a red error marker to the left of the respective line and to the right in the overview-bar of the editor.

### Parse Tree

You can actually take a look at how the plugin will interpret the syntax of the source code by opening the parse tree via the respective button  in the toolbar.

This will open a window (for some odd reasons it tends to open it in the background of the main window when being opened for the first time) showing the syntax structure of the file. This can be used to understand how operator precedence in SQF works (how ArmA will group the commands, e.g. in which order they are being processed and what return value will be fed into what command).

### Type checking

The plugin does not only perform syntax checking (checking whether the commands have been used in a proper way) but also if the arguments fed into the used commands actually fit the command's expectations. Therefore the plugin will issue an error for statements like `hint 3` because `hint` expects a String or formatted text as its argument and not a number.

While ArmA would throw a “Generic error in expression” at you SQDev gently issues an error telling you that you inputted a Number where a String was expected (of course while marking the error visually in the editor as well).

This check is even done syntax-specific as some commands have multiple syntaxes and each one of them uses different types for the second argument depending on what type the first argument was (in what syntax the command is being used).

A good example for this is the `find` command. it can either be used to find an element in an array via the syntax `<Array> find <Anything>` or to find something in a String in which case the syntax is `<String> find <String>`.

As you can see the allowed second argument's type depends on the type of the first argument used. The plugin is capable of detecting and resolving this dependency.

### Limitations

The type-checking has the following limitations:

- At the moment the type of variables is not being tracked. Therefore no errors will be produced when using variables as an argument
- Macros are not being expanded. Therefore they are always considered a valid argument
- If the argument of a command is an array the plugin will not check whether the array contains the proper elements (neither count nor type). This is because array contents are not uniformly documented in the wiki which makes it extremely difficult to automatically extract the needed information.
- Scoping is not implemented yet
- local variables inherited from a calling script are not recognized (and probably never will be because in order to detect those the plugin needed to actually run the code - The same goes for some other SQF-specific-hacking-stuff).

### 4.1.2 StringTableEditor

The StringTableEditor is an editor that is designed for editing and manipulating StringTables (who'd guessed that?). The real feature is that not only does it provide a text-editor for manually editing the StringTable but also a GUI that allows you to never see the actual XML code of the StringTable.

### 4.1.3 RPT-Viewer

The plugin includes a custom RPTViewer which is a view by itself and shows you the contents of the `.rpt` files on your machine.

Therefore you don't have to open them in an additional editor and constantly be switching between editors. Instead you can simply use this view and access all relevant information inside Eclipse.

## 4 Feature Overview

It even supports a filter-function that will do its best to remove repeating lines inside the `.rpt` (e.g. the same error message repeating for ten times) so that the respective message will only be displayed once.

Furthermore it can filter out lines starting with a specific prefix (configurable via the preferences - see section 5.1 on page 21).

### Triggering mechanism

Note that in the current version of the plugin the linking mechanism is only triggered when saving an **.sqf-file**. Therefore you have to re-save a **.sqf** if you have made any changes to a different file (e.g. the **description.ext**) in order for the plugin to export your changes to ArmA.

This is a residue from the beginnings of the plugin and will be addressed in one of the next releases.

### Linking direction

You need to be aware that the linking is one-way only. Therefore any changes made directly in the ArmA directory are not overtaken into Eclipse. Actually it is quite likely that these changes get overwritten the next time the linking mechanism is triggered. If you have some files (like the **mission.sqm**) that don't need editing in Eclipse but get changed directly in the ArmA directory instead refer to section 6 on page 29 for further instructions.

### 4.1.4 Wizards

Wizards are a mechanism inside Eclipse that allow for some automation of common tasks. Usually they provide a window in which some details about that task can be set and then the wizard will perform the task automatically for you.

In SQDev the provided wizards are currently all for managing projects and files within these projects. They are available through the context menu in the navigator.

### SQDev Project wizard

This wizard will create a new SQDev-project for you with some default contents and configurations set. This consists in a basic project structure as is found in the most common ArmA projects, some default files (`description.ext` and `mission.sqm`). As you can see the wizard will create a `mission.sqm` for you so there's no need to jump into ArmA first just to create the new mission you are going to work on. All can be done directly from within Eclipse.

Most notably this wizard creates the `link.sqdev` for you which contains the local project settings for the linking of this specific project. These information can be adjusted in the GUI when invoking the wizard so you don't have to worry about writing it in manually.

### Options

---

- **Project name** - Defines the name of the project. In case the project is a mission it also defines the mission's name.
  - **Project type** - Defines the kind of project that is being created. This does mainly influence the default content contained in the generated project. Currently these types are hard-coded and only the options `Mission` and `Mod` are available.
  - **Terrain** - This option is only available for the *Mission* project type. It defines the terrain the mission will be generated on.
  - **Profile** - Defines the author's profile name that is going to be associated with the generated project. It can only be a name of a valid and existing ArmA profile.
  - **Auto export** - Toggles whether the auto-export (Linking - see section 4.1.8) feature is going to be enabled for this project.
  - **Multiplayer** - This option is only available for the *Mission* project type. It toggles whether the generated mission will be a multiplayer mission (which will alter the linking directory accordingly - The mission will be exported in the `mpMissions` directory)
-

### SQDev Project wizard

**Access:** New » SQDev Project or New » Project - » SQDev Project  
**Context:** Navigator

### SQF File wizard

This wizard will create a new **.sqf**-file for you that will already contain a custom header which contains information about the author (profile name as associated with the current project), the enclosing project, the script's name as well as a default template for description, parameter and return value.

#### Options

---

- **File name** - Defines the name of the file that's going to be created.
- 

### SQF File wizard

**Access:** New » SQF File or New » Other » SQF File  
**Context:** Projects and Folders

### StringTable wizard

This wizard will add a new StringTable to your project. This StringTable will be using the recommended XML structure and can be edited with the StringTableEditor (see section 4.1.2).

The created StringTable file will be empty - The default XML structure will be added as soon as you start editing the StringTable with the respective editor.

### StringTable wizard

**Access:** New » StringTable or New » Other » StringTable  
**Context:** Projects and Folders (Only useful when invoked on the project itself)

### Import as SQDev-project

This wizard will allow you to import an already existing project into Eclipse as a SQDev-project. The benefit of using this wizard is that it will create the needed infrastructure (including all necessary configuration files containing the default settings as set in the preferences (see 5.1 on page 21)).

Therefore you can directly continue on your project without having to worry about what configurations have to be made in order for the project to integrate itself into Eclipse.

### Options

---

- **Path** - This defines the path to the project that should be imported. It should point to the project's topmost directory. You can use the **Browse...** button in order to navigate to the respective directory.
- 

Import as SQDev-project	
Access:	<b>Import...</b> » Import as SQDevProject
Context:	Navigator

### Export SQDev-project

This wizard is responsible for exporting the current project **as a mission!**

Its main use is if Linking (see section 4.1.8 on page 19) is disabled and one wants to export the project as a playable mission to a custom path.

### Options

---

- **Destination** - This defines the directory the project should be exported into. This should point to the directory that later should contain the mission folder (not to the mission folder itself). You can use the **Browse...** button in order to navigate to the respective directory.
-

Export SQDev-project	
Access:	Export... » Export SQDevProject
Context:	Navigator

### 4.1.5 Config- and Header-Editor

The plugin does also include a Config-/Header-Editor that can be used in order to edit `.cpp`, `.hpp`, `.ext` and `.sqm` files.

It's mainly a default Eclipse-text-editor that provides basic syntax highlighting and content-assist (mostly only for macros) and provides macro-support.

That means that the editor will show (included) macros in the content-assist and will highlight them respectively.

### 4.1.6 SQDev-File-Editor

For editing the `link.sqdev` (see section 5.2.2 on page 27) the plugin ships a custom editor that provides syntax highlighting and content-assist for the respective keywords.

### 4.1.7 SQDev-Perspective

A custom perspective within Eclipse that optimizes the UI for the work on SQF projects. This includes the opening and closing of some default views as well as a special selection of context-menu entries.

### 4.1.8 Linking

When working that way in Eclipse you are always working on a copy of the actual project (this might not be true for mod-projects) so you might wonder how your



mission will actually get into the ArmA directories so that it can be opened in ArmA.

That is where Linking comes into play. Linking is an algorithm that will export your project into the respective directory (determined by the `link.sqdev` - more info in section 5.2.2 on page 27).

This overcomes the problem that eclipse has its workspace as a separate folder on the disk in which it will manage its projects. Linking will make sure that any changes made in eclipse will be transferred to ArmA without the user having to worry about it.

### 4.1.9 Command-update

The plugin is capable of updating its internal command-list without any need to mess around in its source code. For this you have to hit the Command update button in the misc-preference-category (see section 5.1 on page 21).

It will then contact all respective sites and update the information about all known commands (this includes the addition of newly introduced commands).

If you ever experience that the plugin does not know a certain command or has outdated command information (e.g. missing a newly introduced syntax) make sure that the needed information is documented in the BIKI (see section 7.3 on page 31) and then invoke the update-mechanism.

This will then take a while (a few minutes) until all sites have been contacted. After that all editors need to be restarted (should be done automatically) and the respective information should be present.

## 4.2 Future Plans

Coming soon

# 5 Configuring the plugin

## 5.1 Settings

The settings of the plugin can be accessed via `Window > Preferences > SQDev`. If expanded there are several more preference categories underneath the `SQDev`-node. For all of those categories the default values can be restored by using the `Restore Defaults` button. Loading the default value for only one of the fields is currently impossible.

If you are not sure what a certain preference is for you can either have a look in the following sections for detailed descriptions or you can use the tooltip every preference-field of the plugin has to get a brief info about it.

### SQDev

---

- **Create Dummy ArmA-infrastructure** - When clicking on this button the plugin will create some dummy-directories and -files mimicking the ArmA directory structure (as needed by the plugin to not complain). These directories will be created in your home directory inside a folder `.ArmaDummyFiles`.

It will then set the values of `Program`, `Documents` and `RPTs` to the respective sub-dummy-folders. Furthermore it will switch off `Default auto-export`.

This option is intended for users that want to use the plugin on a machine that doesn't have ArmA installed on it.

- **Program** - This sets the path to the ArmA installation-folder. The installation-folder is the one that actually contains the `arma3.exe` (or `arma3.i386` on Linux) executable. Note that the plugin will verify that the respective executable is present in that directory!

### Usage of the installation-folder

At this time the installation-folder is not being used for anything but in the future it is planned to use it as a starting point for Mod-lookups.

- **Documents** - This is the path to the directory that contains the **Arma 3** (and **Arma 3 - Other profiles**) directory. On windows this normally is your normal Documents-directory and on Linux it normally is `$HOME$ ▶ .local ▶ share ▶ bohemiainteractive ▶ arma3 ▶ GameDocuments`.

These directories are needed in order to find all profiles of the current user and for determining where the auto-export should point to (it will always point either to the **missions** or **mpMissions** directory so that the respective missions/projects are found by ArmA).

- **RPTs** - This is the path to the place ArmA dumps its log-files (the RPT-files).
- **Default auto-export** - Indicates whether the auto-export flag should be turned on by default when creating a new project or importing one.
- **Default terrain** - Sets the default terrain that should be used when creating a new mission (project).
- **Default profile** - Lets you choose the default profile to use when creating a new project or importing one. Only profiles that are existent on that particular ArmA-installation (see **Documents**) are available to choose from.

---

All of the latter 3 “default-options” can be set for each project individually. When creating a new project you can make the project-specific settings via the project-wizard. When importing a project the default values are being used without asking.

However these options are all ultimately determined by the content of the `link.sqdev` (see section 5.2.2 on page 27) so that is the place these options can always be changed manually for each project.

### Editor

---

- **Enable current line highlighting** - Indicates whether the line the cursor currently is in should be highlighted in the specified color.
  - **Enable bracket highlighting** - Indicates whether matching brackets should be highlighted when the cursor is next to them (if the cursor is next to a  (does also apply for all other brackets) will be highlighted in the source code if this is switched on).
  - **Enable autoComplete** - Indicates whether auto-complete is switched on or off. If auto-complete is on and the content-assist is being invoked in the editor and there is only one possibility to complete the current word, the content-assist won't be opened and the respective completed word is being inserted into the source code without prompting the user.
  - **Parse delay** - This indicates the time in ms between the last entered character into the source code and the invoking of the respective parser.
  - **Colors** - This set of options allow the user to customize all highlighting colors used within the editors. In order to set a new color you simply have to click on the respective colored rectangle, choose a new color in the color-picker and hit select. If the desired color is not directly available in the color-picker you can always use the  button in order to choose an arbitrary color.
- 

### Linking

---

- **Auto clean** - When exporting a project/mission the target directory needs to be cleaned first in most cases (considering the export happens during a project's auto-export). If this option is set to true all of the directory's content will be permanently deleted (except the files set in the `link.sqdev` (see section 5.2.2 on page 27) without asking for permission.

If this is set to false the user will be prompted every time the plugin thinks such a cleaning-process is necessary.

---

### Misc

---

- **Update Command** - When being clicked this will update the internal command-list of the plugin. This includes new commands, syntaxes, descriptions, notes and so on. basically everything the wiki pages of the respective commands have to offer.
  - **Reset Keywords** - After having updated the commands at least once this button allows you to revert the changes made by the last update (in case something went wrong or you're otherwise unsatisfied with the result) and switch back to the version before the update. Note that this can only be done for the version directly before the update. any further changes have to be done manually (see section 5.2.3 on page 28).
  - **BIKI API** - This option does specify the URL to the API of the Bohemia Interactive wiki (BIKI). This is needed for the plugin's command-update-functionality. This option is present so that this parameter can be changed should the address ever change and I'm no longer maintaining the plugin. Therefore you shouldn't mess around with this option unless there is a real need for it.
  - **Main Page Name** - Defines the name of the web-page that lists all available commands. This is also used during the plugin's update-mechanism. Therefore the same rules apply.
  - **Always save on exit** - When set to true (selected) a little pop-up will appear when clicking  or  if there are unsaved changes made to SQDev-preferences. It will ask you whether to save or discard these changes.
  - **Validate deletions** - When checked the plugin will re-validate each deletion triggered by the user instead of simply deleting the respective file(s).
- 

### Views

---

- **RPTs** - This is the path to the place ArmA dumps its log-files (the RPT-files).
- **Format RPTs** - Indicates whether the formatting of the RPT-files displayed

in the RPT-viewer should be turned on (Can also be configured in the view itself).

- **Format prefixes** - When clicked this will open a dialog allowing you to manage the prefixes of lines that should get removed from the displayed RPT when formatting is turned on.
  - **Max blank lines** - Sets the maximum amount of blank lines in a row a displayed RPT should contain when being formatted. All following empty lines will be removed during formatting.
- 

## 5.2 Configuration Files

The behavior of the plugin is dependent not only on the preferences discussed in section 5.1 on page 21 but also on some specific configuration files. This allows a per-project-configuration (useful if you need one project to behave differently than another) and manual adjustments in cases one needs to fine-tune the behavior in a way the plugin won't let you do directly.

### 5.2.1 SQDev-files

SQDev-files are a new file-format being introduced in the plugin. They are used to configure the plugin's behavior in a local scope (e.g. on a per-project-basis).

They can provide information via so-called **attributes** and **annotations** (together referred to as “information”) and follow the following set of (syntactical) rules:

- Every information has to end with a newline
- No more than one information must be specified per line
- No leading whitespace in front of an information is allowed (apart from newlines)
- Blank lines are allowed

## 5 Configuring the plugin

- Whitespace accounts for line-content (a line containing only a blank is not considered a blank line)
- Comment-lines (starting with `///`) are supported
- Each comment has to be on a separate line (may be indented by whitespace)
- If a line is not empty it is expected to be a comment or a information
- Attributes may only be set once in the file
- Annotations can be used arbitrarily often
- SQDev-files are case-sensitive

An attribute is a key-value pair in the form `<key>=<value>;` (Note the mandatory trailing semicolon). Blanks and tabs are allowed between `<key>` and `=` as well as between `=` and `<value>`. Any blanks or tabs between `<value>` and `;` will be considered part of `<value>`.

An annotation does also operate on a key-value basis and has the format `@<key> "<value>"` (Note the absence of a trailing semicolon). The key must be followed by a blank which may be followed by one or more blanks and tabs before the value-string (which needs to be enclosed in double-quotation-marks).

The difference between attributes and annotations is that attributes set a variable (thus they must only be specified once in the file) whereas annotations will be added to a internal list (Therefore using an annotation is the equivalent to a add-Function on the list).

### Seemingly arbitrary syntax

When reading the syntax-specifications for SQDev-files one might get the impression that they are kind of arbitrary and unnecessarily restrictive. That is absolutely correct! This syntax is a residue from the early days of the plugin and is a result of a lazy file-"parsing"-strategy.

Changing the syntax is somewhere on the ToDo-list but doesn't have any real priority because it is seldom needed to insert new information into the file. And changing existing information (not needed very often either) doesn't require most of those rules.

### 5.2.2 link.sqdev

The `link.sqdev` is a special file in each SQDev-project containing information needed during the linking process (see section 4.1.8 on page 19).

It is the (at the moment) only “SQDev-file” (see section 5.2.1) the plugin uses. It offers the following information-specifications:

#### Attributes

---

- **autoExport** - Indicates whether auto-export (see section 4.1.8 on page 19) is enabled for this project. Possible values are `true` or `false`
  - **exportDirectory** - This is the directory the mission is going to be auto-exported into (if activated). During the initial creation this path will be sensitive to the set profile-attribute.
  - **terrain** - Sets the terrain the mission will be played on.
  - **profile** - Sets the profile associated with this project. This does influence the author-field in newly created `.sqf` files.
- 

#### Annotations

---

- **ignore** - All files or folders (characterized by their relative path from the project’s root) specified with this annotation are being ignored during auto-export of the project. If a folder is specified all of its content is ignored as well.
  - **preserve** - All files or folders (characterized by their relative path from the project’s root) specified with this annotation are being preserved during the cleaning process performed before each auto-export. Effectively this prevents the respective files/folders from being deleted in the target directory of the auto-export. If a folder is specified all its content is preserved as well.
-



### 5.2.3 SQFKeywords.txt

This file contains the information about all available commands as can be found in the BIKI. It uses a XML-ish syntax.

It can be found inside your Eclipse workspace (the one assigned by yourself during the initial Eclipse-start) under `.metadata ▸ .plugins ▸ raven.sqdev.misc ▸ resources`.

Changes made to this file will affect the way the plugin treats commands. It determines what keywords correspond to commands, what syntax they can be used with and so on. Before editing it you should always make a backup of the file as manual edits are not always as trivial as one might think (This is another point somewhere on the ToDo-list).

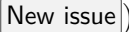
The good news is that the plugin is designed in a way that this file does not require any manual editing unless the update-mechanism breaks.


## 6 Tips and Tricks

Coming soon

# 7 Contributing

## 7.1 Reporting Bugs

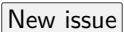
If you encounter any problems with the plugin please take the time to report a bug-report on GitHub (<https://github.com/Krzmbzrl/SQDev/issues> - then hit ).

Describe as best as you can what happened and don't forget to include a Plugin-Info. In order to create one you have to hit the  button in the toolbar. This will create a `.zip` file containing information about your Eclipse-installation as well as some plugin-specific information that will help me determining where the problem may come from and thus providing faster support without having to ask the same questions for every report.

If you are unsure whether the discovered issue is an actual bug file the bug-report nonetheless. Chances are high that it is one and even if it's not: The fact that you found it weird enough to consider it a bug is reason enough to draw my attention to this issue.

## 7.2 Feature requests + Improvement suggestions

If you are using the plugin and realize that there is a feature missing you'd like to see included or you have a (neat) idea to improve an existing feature you can file a feature request / improvement suggestion.

For that visit <https://github.com/Krzmbzrl/SQDev/issues> and hit the  button in the top right corner.

Please be as detailed as possible when describing the wished feature/improvement.

If you are not really sure about what exactly it is that you want I'd ask you to join the official Discord server (see section 7.4) and post it there. Then it can be properly discussed until a clear concept is being achieved.

I will then review the issue and determine whether this is doable or not. Don't be shy about such requests. No offense is taken in over-eagerly feature-requests or improvement suggestions (even if they'd imply to do something completely different) as long as you're presenting it in an appropriate way (constructive).

With these requests the same principle applies as with bug-reports: It is better to have too many than too few.

### 7.3 Maintain the BIKI

By maintaining and improving the Bohemia Interactive wiki (short: BIKI) you are also contributing to the plugin as it receives its information about all available commands from there.

Therefore you can make sure that these pages include all information about the commands that someone might find useful. If you discover a property of a command that is not yet listed there, add a note so that following visitors will find it there. Furthermore the plugin will show this note after a command-update (see section 4.1.9 on page 20) directly in the editor when accessing the respective command's information.

As the plugin as a program can only be so smart when gathering information in the BIKI it needs the syntax of a command to be properly formatted. It is important that each parameter-placeholder in the syntax is properly resolved underneath it. Properly means:

- The placeholders have to be spelled the same in the syntax and in the resolution underneath
- In the resolution each placeholder has to be followed by a colon followed by the type this placeholder needs to be
- Additional description of a placeholder needs to be separated by a - from the placeholders type (and it needs to be specified after the type)

## 7 Contributing

- Optional parameter have to be specified after the mandatory ones
- Optional parameters are indicated by the “(optional)” flag directly after the placeholder-resolution and before the colon (see <https://community.bistudio.com/wiki/addAction>) for examples

If you see one of those rules violated it is likely that this page will break the plugin’s update-mechanism and if you format the site properly you directly contributed to making the update-mechanism more stable and reliable.

### 7.4 Contact

You can contact me on the official Discord server (<https://discordapp.com/invite/Ny3a4QS>). I will be more than happy to answer any question about the plugin and anything related to it (e.g. Eclipse).

## 8 Version History

### 0.7.4

- | Improved: Switched to latest ANTLR release (4.7) which should decrease the parse-time
- | Fixed: Continuous NullPointerExceptions during normal coding
- | Fixed: Building projects incredibly slow
- | Fixed: Increasing command-access time during validation in builder
- | Fixed: Permission problems when creating dummy directories
- | Fixed: Missing return value after project building

### 0.7.3

- | Added: Parser testcases for improved stability throughout development
- | Added: Possibility to create necessary file-  
infrastructure as dummy on machines that don't have ArmA installed
- | Added: MP option to Project wizard
- | Added: Global Project Parsing
- | Added: SQDev Project Nature for identifying SQDev projects properly
- | Added: Custom SQDev Navigator (experimental only)
- | Improved: SQDev projects are now encoded with UTF-8 per default (independently from the system's default)
- | Improved: Command info is now two staged (first basic, then full)
- | Improved: Editors now restart automatically after command update
- | Improved: Command update now uses the BIKI API
- | Improved: SQF validation is no longer bound to the SQFEditor allowing for more flexible usage

## 8 Version History

- | Improved: Hidden files are now ignored during export
- | Improved: All parsing is now done with the SLL(\*) approach
- | Fixed: Little bugs in command update
- | Fixed: Error markers are now displayed on external files as well
- | Fixed: (Un)Indentation of whole selection area is no longer broken

### 0.7.2

- | Added: Rudimentary support for mod projects (creation + import)
- | Fixed: Wrongly escaped quotation marks in config files
- | Fixed: Binary operators that don't have (yet) a right argument causing a NullPointerException during parsing
- | Fixed: Error occurring on startup of the StringTableEditor
- | Fixed: Inappropriate selection in the package-tree of the StringTableEditor
- | Fixed: Error during command update on windows machines
- | Fixed: RPTViewer formatting now works as expected
- | Fixed: Error markers only appearing at the start of an element instead of underlining it completely
- | Fixed: Endless loop in SQDev-Editor when typing in a '@'
- | Improved: Numbers between 0 and 1 can now be entered as '.x' (instead of '0.x') without getting an error
- | Improved: Return value is now determined according to the context in which the operator is used
- | Improved: Operators having multiple return values depending on the used syntax are now respected
- | Improved: Parsing now gets triggered by content change instead of keyboard inputs
- | Improved: Vital commands (control structures) can no longer be left out

### 0.7.1

- | Added: Preference specifying whether a notification should pop up on parseMode change
- | Improved: Includes now check for the use of a backslash
- | Improved: SQF parser now recognizes numbers in scientific and hexadecimal notation

- | Fixed: Parsing getting really slow under certain circumstances
- | Fixed: Files do no longer get parsed multiple times at once
- | Fixed: Includes on unix systems no longer fail because of backslash
- | Fixed: Multiline comments not being highlighted immediately

### 0.7.0 - Syntax Check

- | Added: SQF syntax checking
- | Added: Support of standard magic variables (e.g. `"_this"`)
- | Added: Highlighting for macros
- | Added: Config editors
- | Added: Plugin info creator
- | Added: FileSystem listening framework
- | Added: SQDev perspective
- | Added: RPTViewer as a new View
- | Added: Preference for the RPT location
- | Added: Preference for user validation of deletions
- | Added: Preferences for formatting RPT content
- | Added: New preference editor for multiple Strings
- | Rewritten: SQF grammar
- | Rewritten: Preprocessor parser
- | Improved: Variable declarations via `"params"` and `"private"` are now recognized
- | Improved: Implicit variable declarations in a for loop are now recognized
- | Improved: No code folding markers for code that is written in only one line anymore
- | Improved: Operators like `"+"`, `"-"`, etc. are also included in a command update
- | Improved: Search Navigator-view for current selection as well
- | Improved: Errors on Syntax-Parsing during KeywordCollection of commands can now recover
- | Improved: SQDevInfobox is now persistent and is not discarded when eclipse is not in focus
- | Improved: Localisation of directories now supports linux
- | Improved: After creating a new SQDev project the user is prompt whether the SQDev prespective should be opened



## 8 Version History

- | Improved: Parsing is done in an extra thread
- | Changed: Changed text of button in misc-preferences: old: "Update keywords", new: "Update commands"
- | Changed: Relocated miscellaneous preferences into the misc preference page
- | Changed: Parse delay preference is now specified in milliseconds for better adjustments
- | Changed: Syntax of the SQFKeywords-file is now XML-like to increase readability
- | Fixed: Comment at the end of the file marked as an error
- | Fixed: Nested macros are marked as an error
- | Fixed: ArmA files get properly located on linux
- | Fixed: Minor Bugs in IntegerPreferenceEditor
- | Fixed: Bug causing crash on linux on profile-retrieval
- | Fixed: Bug occuring when opening two SQDevInfoboxes simultaneously
- | Fixed: File->Open file ... does no longer result in a crashed editor
- | Fixed: ConcurrentModificationException during saving
- | Fixed: Code folding markers are no longer "persistent"
- | Fixed: Various minor bugs in the SQF command collection
- | Fixed: Information hover only appearing if the operator is written in the "proper" way
- | Fixed: Open wiki page for commands sometimes lead to complete crash
- | Fixed: Empty hover info appearing when hovering over macros/variables

### 0.6.1

- | Added: Tanoa as terrain dropdown
- | Added: Macro-support
- | Added: Macro (syntax) check
- | Fixed: save problem caused by code folding [need Info]
- | Fixed: StringtableEditor not overtaking changes from the UI into the XML
- | Fixed: Error when opening SQDevFileEditor
- | Fixed: SQFParser no longer bails out with macros
- | Fixed: Bug in CommandUpdate preventing it from processing a few commmands (e.g. safeZoneY)
- | Fixed: Bug in syntax processing
- | Improved: StringtableEditor does no longer auto-collapse

## 8 Version History

- the tree in the UI
- | Improved: Languages columns in StringTableEditor now get sorted -> no arbitrary order anymore
- | Improved: CommandUpdate can now recover on error by retry or skip
- | Changed: Enabled error markers in SQF-editor

### 0.6.0

- | Improved: Initial script header (added 2 trailing newLines)
- | Improved: SQF icon
- | Changed: All editors should now save in UTF-8 encoding
- | Added: Syntax attribute for commands
- | Added: Parsing for SQF files
- | Added: Parsing for Stringtable.xml files
- | Added: Stringtable editor
- | Added: ParseDelay preference
- | Added: Multi color syntax highlighting for keywords
- | Added: Option to reset keywords to last backup
- | Added: Code folding

### 0.5.3

- | Fixed: Bug preventing you from exporting empty files
- | Fixed: Bug allowing you to give invalid names to projects /scripts
- | Changed: Initial content is now description header instead of scopeName
- | Changed: CharacterCompletion improved (disabled in Strings and in front of words)

### 0.5.2

- | Fixed: Bug preventing you from creating a new SQF file when  
using the project explorer

### 0.5.1

- | Fixed: Bug preventing you from creating a new SQF file
- | Fixed: Bug preventing the SQF editor from working properly  
on first startup

### **0.5.0**

- | Added: Content assist
- | Added: preference to enable/disable auto completion
- | Added: Resource framework
- | Added: Keyword update function
- | Added: Additional info display
- | Added: Keyword–Wiki–Connection
- | Added: Hover Assist

### **0.4.0 - Linking-Update**

- | Changed: A SQDevProject now contains a few more files (`description.ext`, `link.sqdev`)
- | Changed: Working tasks now run in different thread
- | Changed: SQF keywords are now case insensitive
- | Changed: SQDevProjects create a `mission.sqm` file
- | Changed: Created files are now in windows format ("`\r\n`" instead of "`\n`")
- | Added: SQDevFile fileType
- | Added: Linking preference category
- | Added: Auto clean preference
- | Added: Project wizard now has tooltips for every option
- | Added: Import and Export wizards
- | Added: More options when creating a SQDevProject
- | Added: SQDev editor
- | Fixed: Bug in the LayoutManager for the SQDevPreferencePage
- | Fixed: Bug preventing a fileCreation in subFolders
- | Fixed: Bug causing word parts getting highlighted as a keyword
- | Fixed: Bug: Strings that end with a quotation mark are not colored properly

### **0.3.0 - Preference-Update**

- | Added preferences
  - Added preference page
  - Added preference for specifying the Arma folders (in documents and programs directory)
  - Added preference for setting the behaviour on closing the preference page when there are some unsaved

## 8 *Version History*

preferences

- | Changed: Editor behaviour now specified by preferences
- | Fixed: version number is now correct