

Structure101g

ActionScript Flavor

Inhalt

Synopsis	4
Installation.....	5
Installation of (Re)Structure101g	5
Installation of the ActionScript flavor	5
Usage	6
Analyzing a Flash Builder Project.....	7
Analyzing a Flexmojos Project.....	8
Analyzing a Custom Project Structure.....	10
Using a Compiler Configuration File.....	10
Running the Analyzer from the Command Line	11
Running the Analyzer using Ant	13
Dealing with Analyzer Errors	14

Synopsis

In the Java world Structure101 is a well-known tool for performing a structural analysis of Source Code. With Structure101g [Headway Software](#) delivered a by-product that allows analyzing source code of any kind with the use of so called flavors. Structure101g offers almost all the functionality known from its Java counterpart.

Together with [Headway Software](#), [Catalysts](#) created such a flavor for the ActionScript language, which allows analyzing arbitrary Air-, Flex-, or ActionScript projects. Besides the usual Flex Builder projects the ActionScript flavor also supports IntelliJ IDEA and flex-mojos project environments.

This flavor mainly uses the Open Source library [Asycle](#) for the structural analysis of the ActionScript codebase, and as such it relies on the capabilities of that library. At the time of writing this means that the flavor is restricted to:

- **Flex SDK 3.x (tested with 3.3, 3.4, 3.4.1, 3.5)**
- **Flex SDK 4.x (tested with 4.0, 4.1, 4.5)**
- **Java 1.5+**
- **Flash Builder projects only, Adobe Flash Professional projects are not yet supported**

The flavor works on any operating system that is supported by Structure101g, for more information take a look at <http://www.headwaysoftware.com/products/languages.php#more-section>.

Installation

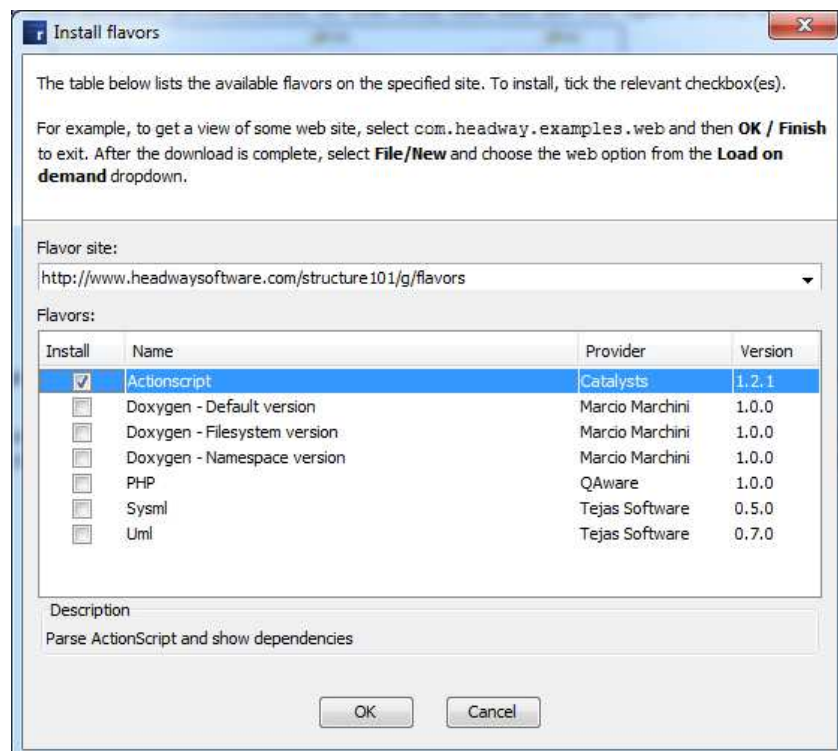
First you have to go to <http://www.headwaysoftware.com/products/> and decide on one of the available products there.

Installation of (Re)Structure101g

1. download the appropriate version of (Re)Structure101g from the Headway Software downloads page at <http://www.headwaysoftware.com/downloads>
2. run the (Re)Structure101g installer and follow the installation instructions
3. after completing the installation run (Re)Structure101g

Installation of the ActionScript flavor

1. after starting (Re)Structure101g select „Flavors“ – „Install...“ from the menu, the „Install Flavors“ screen appears
2. select the ActionScript flavor in the list of available flavors and confirm your selection by pressing the "OK" button
3. the flavor is now installed



Usage

Based on the fact that there are so many different SDK versions, IDEs, and Tools we have learned that in some circumstances it can be particularly hard to get up and running with your project.

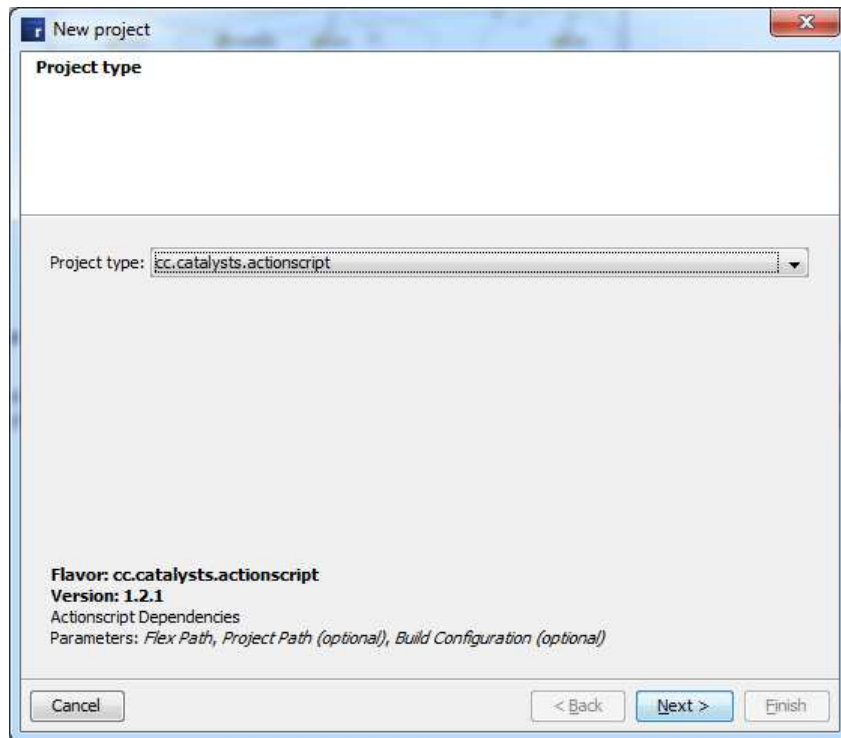
The flavor mechanism in Structure101g basically involves three steps. In the first step all the parameters necessary to perform the structural analysis are collected by the “New project” wizard. After completing the wizard the actual analyzer, which is part of the flavor, is invoked. The analyzer produces a raw dependency graph that is handed back to Structure101g, which in the final step walks that graph and does all the structural analysis (i.e. finding tangles) on it.

We decided to make the wizard in step one as simple as possible, just enough to cover the most general cases to make the majority of the users happy. If it happens to be that you are part of the illustrious circle of people with a reasonable large codebase or if you are using rather uncommon configuration options the wizard might possibly not have collected enough information for the analyzer to do all its magic. We concluded that you will get the most freedom if we just break that cycle here and offer a command line interface to the analyzer as well so that you can tune every single parameter just like as you would with the compiler. As a neat side effect that enables you to offload the most time consuming of the three steps, the analysis, to your continuous integration server and thus have the analysis available for every revision of your precious project.

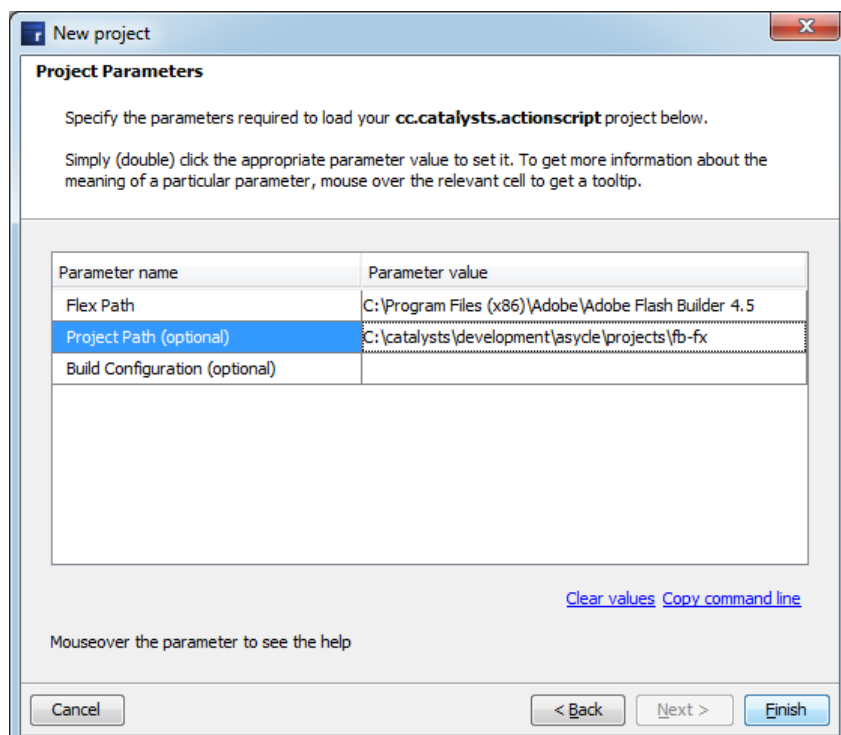
For detailed instructions on how to use the ActionScript flavor, we additionally recommend watching the screencast on <http://www.catalysts.cc/products/structure101g-actionscript/>.

Analyzing a Flash Builder Project

If you are using Flash Builder for Flex development and manage all your dependencies and compiler configuration within your Flash Builder project, just open a new project and select the ActionScript project type.



On the next wizard page select the path to either your Flash Builder installation folder or, if you have downloaded it separately, to the location of the Flex SDK.



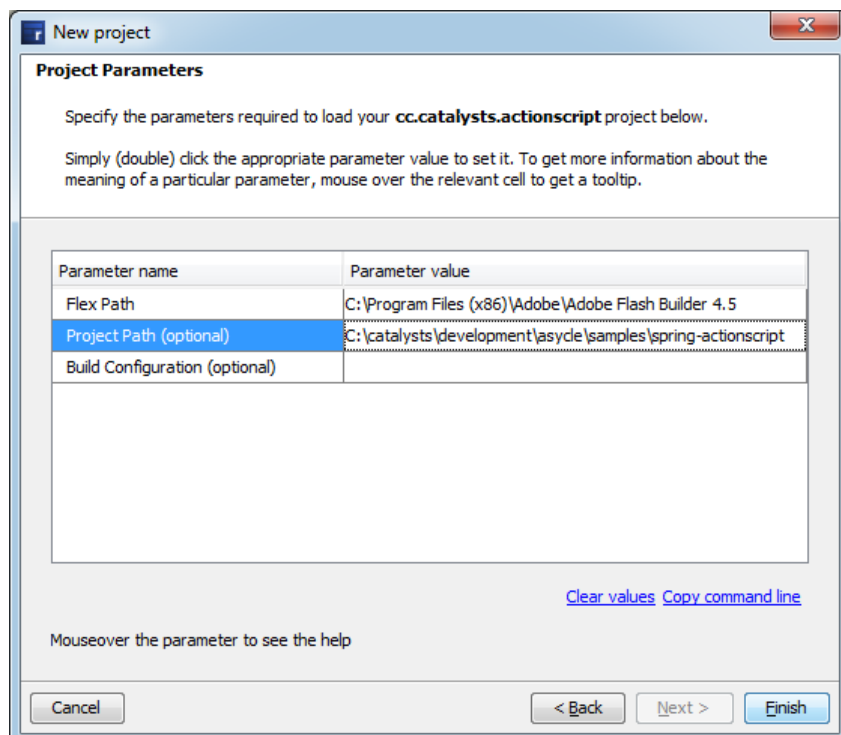
If you press “Finish” the analyzer will start examining your project structure and open up the dependency graph in Structure101g.

Analyzing a Flexmojos Project

If you prefer managing your build configuration with maven rather than with Flash Builder you might be happy to hear that Flexmojos projects can also be opened directly from within the “New project” wizard.

Mind that maven itself is not part of the flavor installation. In order to be able to open Flexmojos projects, maven has to be present on your computer and the path to your maven installation has to be exported to your environment as M2_HOME (or MVN_HOME, MAVEN_HOME, M3_HOME – to cover most of those possible derivatives).

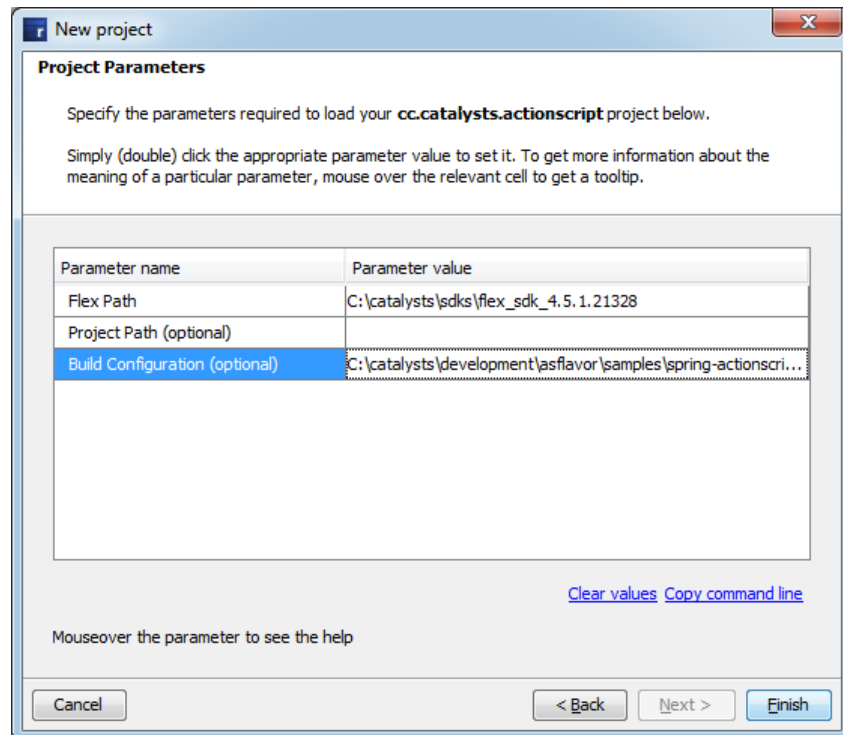
Other than that, just point the “New project” wizard to the directory that contains your projects pom.xml file. Note that you can only analyze individual modules, it is not possible to just point the wizard to the location of a super-pom in order to see the structure of all modules at once. Nevertheless, if you analyze a single module you do see the outgoing references to other modules as library dependencies. Since there cannot be any tangles in module dependencies anyways you don’t lose any dependency information by this limitation.



The Flexmojos integration strongly relies on maven and Flexmojos itself and can require some experience in using those tools in order to sort out possible issues.

Technically, we are invoking “flexmojos: generate-config-swc” and “flexmojos: generate-config-swf” respectively to let maven dump the compiler configuration. Pointing the “New project” wizard to a

directory with a pom.xml hence is just a shortcut to invoking the maven targets manually and pointing the “New project” wizard to the generated build configuration instead of the project path.



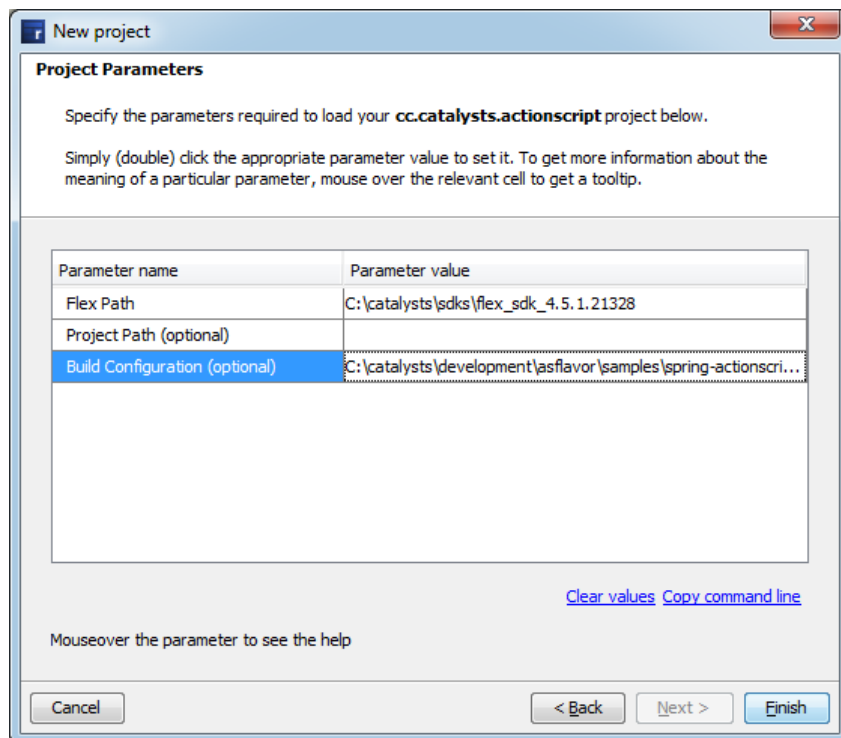
Whenever you experience any problems analyzing your Flexmojos project you can try performing those two steps manually. Most of the times the problems are maven related and the generation of the build configuration fails for some reason. If you cannot generate it manually, neither can Structure101g.

Analyzing a Custom Project Structure

If you prefer being in control of everything and usually build your sources from the command line or with tools like ant, you might find it relieving that Structure101g is trying to preserve that freedom.

Using a Compiler Configuration File

The simplest and probably most versatile way to do so is by creating an XML compiler configuration (http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf69084-7fca.html) which can be used by all the SDK tools, the ant tasks, and Structure101g. Just point the “New project” wizard to your XML file.



Running the Analyzer from the Command Line

If the compiler configuration file alone doesn't provide you with the necessary flexibility or if you want to integrate the analysis into your build process you might consider running the analyzer from the command line.

The analyzer itself is a patched version of the Flex SDKs compc compiler, that basically performs most of the compilation steps of the actual compiler but instead of emitting bytecode it dumps the dependency structure as an XML file. As such, it understands the exact same command line parameters as compc would, including a few additional ones:

- api** takes the class name of the compiler API bootstrapper. Valid values are flex2.compiler.Flex3Api, flex2.compiler.Flex4Api, and flex2.compiler.Flex45Api
- analyzer** path to the jar file containing the actual Structure101g specific outputter. The core of the analyzer is written in a way that the dependencies could be dumped in different formats.

The usage of the analyzer is best described in a short example, so lets take a look at analyzing the flexlib open-source project. First you need to checkout the sources by running:

```
svn checkout http://flexlib.googlecode.com/svn/trunk/ flexlib
```

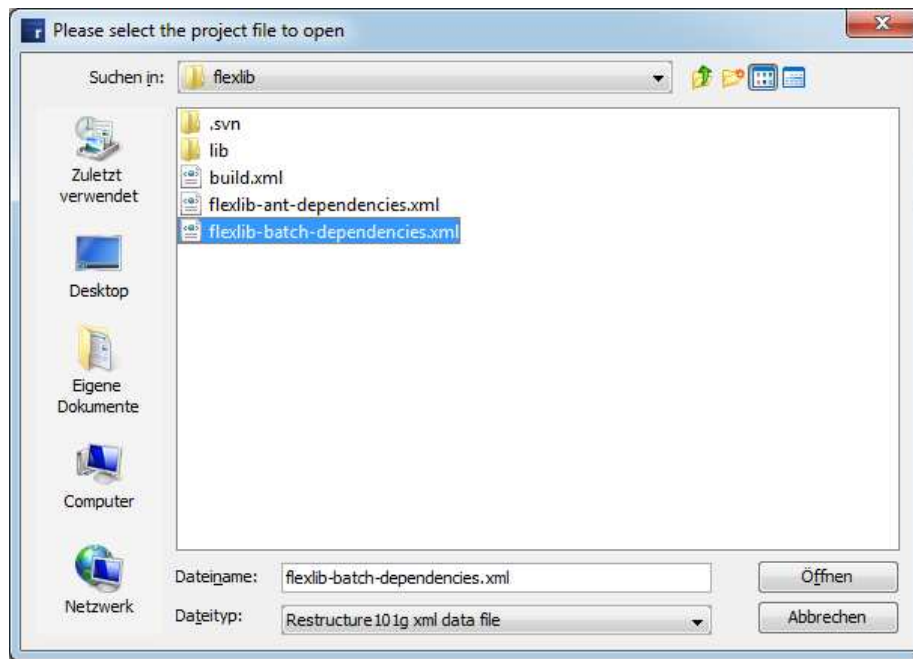
After downloading the sources you can invoke the analyzer by running the following batch script:

```
set FLEX_HOME=<path to flex sdk>
set FLAVOR_HOME=<path to flavor>
set PROJECT_HOME=<path to flexlib folder>

rem because of some classloader hacks within the flex SDK those jars need to be copied to the
SDK directory first
copy lib\asycle.jar %FLEX_HOME%\lib >NUL

rem run the flavor
java -jar "%FLEX_HOME%\lib\asycle.jar" +flexlib="%FLEX_HOME%\frameworks"
-api=flex2.compiler.Flex4Api -analyzer="%FLAVOR_HOME%\lib\actionscript.jar"
-define=FLEX_TARGET_VERSION::flex3,false -define=FLEX_TARGET_VERSION::flex4,true
-source-path="%PROJECT_HOME%\src" -include-sources="%PROJECT_HOME%\src"
-output="%FLAVOR_HOME%\flexlib-dependencies.xml"
```

After running the analyzer you can then just open the resulting flexlib-dependencies.xml file in Structure101g by going to File – Open...



Note that you have to change the file type to “(Re)Structure101g xml data file” in order to be able to pick XML files.

Running the Analyzer using Ant

Just like you can invoke the analyzer from the command line you can run it as an Ant task and hence directly integrate it into your build process.

Here you can see the previous example as an Ant build script:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="asycle" default="report">
  <property name="flex.home" value="<path to flex sdk>"/>
  <property name="flavor.home" value="<path to flavor>"/>
  <property name="project.home" value="<path to flexlib folder>"/>

  <path id="ant.classpath">
    <fileset dir="${flex.home}/ant/lib">
      <include name="*.jar"/>
    </fileset>
    <fileset dir="${flavor.home}/lib">
      <include name="*.jar"/>
    </fileset>
  </path>

  <taskdef name="asycle" classname="org.fluffnstuff.asycle.ant.AsycleTask"
    classpathref="ant.classpath"/>

  <target name="report">
    <property name="FLEX_HOME" value="${flex.home}"/>

    <!-- this file needs to be there to pick up jars from the flex sdk -->
    <copy todir="${flex.home}/lib">
      <fileset dir="${flavor.home}/lib">
        <include name="asycle.jar"/>
      </fileset>
    </copy>

    <asycle fork="true" output="${flavor.home}/flexlib-dependencies.xml">
      <jvmarg line="-Xmx512M -Xms256M"/>

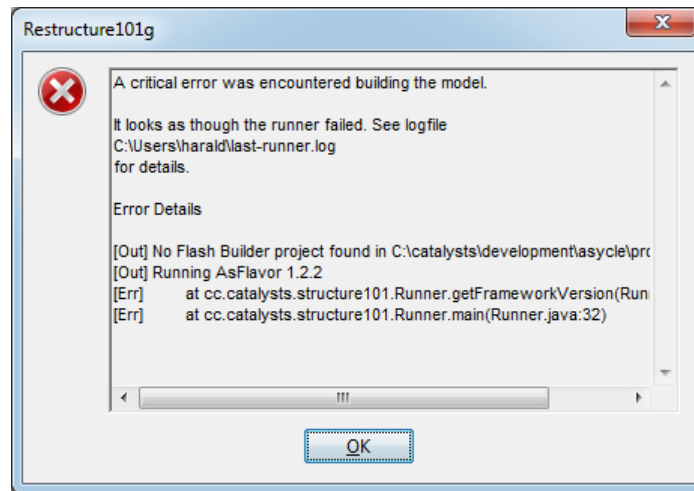
      <api class="flex2.compiler.Flex4Api"/>
      <analyzer filename="${flavor.home}/lib/actionscrip.jar"/>

      <define name="FLEX_TARGET_VERSION::flex3" value="false"/>
      <define name="FLEX_TARGET_VERSION::flex4" value="true"/>

      <source-path path-element="${project.home}/src"/>
      <include-sources dir="${project.home}" includes="src"/>
    </asycle>
  </target>
</project>
```

Dealing with Analyzer Errors

In case something goes wrong, e.g. if you point Structure101g to a project path that does not contain a valid Flash Builder project, you will see an error window presenting the actual output of the analyzer.



Most of the times the error message should give you an idea what probably could have gone wrong, in some cases the error message can be more cryptic though. In that case don't hesitate to contact our support, but ensure that you attach the whole output to your support request, and not just a screenshot of the visible area.