



## **Quick-start Guide**

# Set-up

## Installation

For installation, follow instructions found here:  
<https://github.com/deverl-ide/deverl/blob/master/INSTALL.md>


## Paths

In order for Deverl to function correctly, it must know the paths to the various Erlang binaries. There are three it needs to know; erl (the Erlang runtime system), erlc (the Erlang compiler) and dialyzer (the Erlang static type checker).


In most cases Deverl should automatically discover the correct paths, so the user does not have to do anything at this set-up stage. However, if automatic discovery should fail a dialog will appear notifying the user which binary could not be found. The user must then set the path manually in *Preferences > General*. The preferences dialog can be opened via the Deverl menu bar.

## Creating new files

Files can either belong to a project, in which case they appear in the *src* folder of a project directory, or they can be standalone modules. This can be easily determined from Deverl's file navigation tree on the side bar, which is split into *Projects* and *Standalone Files*.

To create a new file, simply click on the  button or choose *File > New* from the menu. Complete the *New File* dialog and click *Finish*.

## Creating new projects



Deverl has its own project management system which it keeps track of via its system preferences file which can be found in Deverl's ebin folder. To create a new project, click the  icon or choose *File > New Project* from the menu, type the project's name into the *New Project* dialog, and click *Finish*. A project directory structure will be generated in the default Deverl projects folder, and the project can now be opened easily from Deverl's *Open Project* dialog.

The project directory that is created uses standard Erlang project format, which contains the following folders:

<i>src</i>	Contains all Erlang source code
<i>include</i>	Contains the projects header (.hrl) files.
<i>ebin</i>	Contains all compiled Erlang binary files for the project.
<i>priv</i>	Contains all other application specific files.

Existing projects can be imported if they are not already recognised by Deverl. Click *File > Import > Import Erlang Project*. Use the browse dialog to enter the location of the project and click *Import*. It is very important that the project directory you import follows the standard structure as described above.

## Compiling & Running

Single modules can be compiled by clicking the  button and projects can be compiled in full by clicking the  button.

Any compile errors are shown in the *Compiler Output* window next to the shell.

Once a module is compiled successfully it is loaded into the shell ready to use. Once a project is successfully built, all modules in that project are loaded into the shell and can be used.

## Tools

### Writing & Running Tests

To write and run eunit tests, the eunit header file must be included in the module in which the tests are written. Usually Devel1 includes this file automatically as part of the module skeleton at file creation, and takes the form:

```
-include_lib("eunit/include/eunit.hrl").
```

To write a test function, define a function name for the test and then append '\_test' to that name like so:


```
function_name_test() ->
```

The function name can be any atom, but must be appended with '\_test'. After the declaration comes the expected result, and we test this expected result for equality with the actual result. Take for example a function which sums two numbers, a test for this function may be written:

```
sum_test() -> 10 = sum(8,2).
```

So, the structure in which the test must be written is thus:


```
Test Name() -> Expected Result = Actual Result.
```

Once the test is written, click on the  button or choose *Tools > Run Tests* from the menu. This will compile the module, run the tests, and show the test results in the side bar.

### Dialyzer

Dialyzer is an analysis tool which performs type checking, and can detect unreachable code and unnecessary tests. In order for Dialyzer to function it needs a Persistent Lookup Table (PLT). When running dialyzer for the first time, Devel1 will ask to generate a PLT. This is a process that can potentially take a long time, but once done, Dialyzer can be run on any module in a matter of seconds.

In order for Dialyzer to check types in your program, each function must have a specification statement defined. Information on how to define a spec statement can be found here: [http://www.erlang.org/doc/reference\\_manual/typespec.html#id76283](http://www.erlang.org/doc/reference_manual/typespec.html#id76283).

To run Dialyzer click the  button. Select the modules you wish to test and click *Run*. The results of the test will be displayed in the *Log* window at the bottom-left of the IDE.

## **Observer**

To run the Erlang Observer, simply click the  button or choose *Tools > Run Observer* from the menu. Observer is a program that comes bundled with the Erlang runtime system which provides useful data and statistics about Erlang processes.

## **Code Editing**

Editor windows can be split by clicking and dragging on the document's tab. It can be repositioned in the current group or a new splitter window can be created by dragging to other areas of the editor window.