devonfw-ide 3.2.2

The devonfw community

Version 3.2.2, 2019-12-17_14.08.34

Table of Contents

Introduction

devonfw provides a solution to building applications which combine best-in-class frameworks and libraries as well as industry proven practices and code conventions. It massively speeds up development, reduces risks and helps you to deliver better results.

This document contains the instructions for the tool `devonfw-ide` to setup and maintain your development tools including your favorite IDE (integrated development environment).

1. Features

Every developer needs great tools to work efficient. Setting all this up manually can be tedious and error-prone. Further, different projects may require different versions and configurations of such tools. Especially configurations like code-formatters should be consistent within a project to avoid diff-wars.

The `devonfw-ide` will solve all these problems. Here are the features we can offer for you with `devonfw-ide`:

- **Efficient**
  Setup your IDE within minutes tailored for the requirements of your project.
- **Automated**
  Automate the setup and update, avoid manual steps and mistakes.
- **Simple**
  KISS (Keep It Small and Simple), no installers or tool-integrations that break with every release. Instead use archive-files (zip or tar.gz), templates and simple shell scripts.
- **Configurable**
  You can tweak the configuration to your needs. Further the settings contain configuration templates for the different tools (see configurator).
- **Maintainable**
  For your project you should copy these settings to an own `git` repository that can be maintained and updated to manage the tool configurations during the project lifecycle. If you use github or gitlab every developer can easily suggest changes and improvements to these settings via pull/merge requests without ending in chaos with big teams.
- **Customizable**
  You need an additional tool that we never heard of? Put it in the `software` folder of the structure. The `devon` CLI will then automatically add it to your `PATH` variable.
  Further you can create your own commandlet for your additional tool. For closed-source tools you can create your own archive and distribute to your team members as long as you care about the terms and licenses of these tools.
- **Multi-platform**
  Works on all major platforms, which are Windows, Mac, and Linux.
- **Multi-tenancy**
  Have as many instances of the `devonfw-ide` "installed" on your machine for different projects

with different tools, tool versions and configurations. No physical installation and no tweaking of your operating system. "Installations" of `devonfw-ide` do not interfere with each other nor with other installed software.

- **Multiple Workspaces**
  Support working with different workspaces on different branches. Create and update new workspaces with few clicks. See the workspace name in the title-bar of your IDE so you do not get confused and work on the right branch.
- **Free**
  The `devonfw-ide` is free just like everything from devonfw. See TERMS_OF_USE for details.

1.1. IDEs

We support the following IDEs:

- Eclipse
- Visual Studio Code
- IntelliJ

1.2. Platforms

We support the following platforms:

- java (see also devon4j)
- C# (see devon4net)
- node.js and angular (see devon4ng)

1.3. Build-Systems

We support the following build-systems:

- mvn (maven)
- npm
- gradle

However, also other IDEs, platforms, or tools can be easily integrated as commandlet.

1.4. Motivation

`TL;DR`? Lets talk to developers in the proper language. Here are examples with `devonfw-ide`:

```
[/]$ devon
You are not inside a devonfw-ide installation: /
[/]$ cd /projects/devonfw
[devonfw]$ mvn
zsh: command not found: mvn
[devonfw]$ devon
devonfw-ide environment variables have been set for /projects/devonfw in workspace
main
[devonfw]$ mvn -v
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-
24T20:41:47+02:00)
Maven home: /projects/devonfw/software/maven
Java version: 1.8.0_191, vendor: Oracle Corporation, runtime:
/projects/devonfw/software/java
Default locale: en_DE, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.3", arch: "x86_64", family: "mac"
[devonfw]$ cd /projects/ide-test/workspaces/test/my-project
[my-project]$ devon
devonfw-ide environment variables have been set for /projects/ide-test in workspace
main
[my-project]$ mvn -v
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-
24T20:41:47+02:00)
Maven home: /projects/ide-test/software/maven
Java version: 11.0.2, vendor: Oracle Corporation, runtime: /projects/ide-
test/software/jdk/Contents/Home
Default locale: en_DE, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.3", arch: "x86_64", family: "mac"
[ide-test]$ devon eclipse
launching Eclipse for workspace test...
[my-project]$ devon build
[INFO] Scanning for projects...
...
[INFO] BUILD SUCCESS
```

This was just a very simple demo of what `devonfw-ide` can do. For further details have a look at our CLI documentation.

Now you might ask:

- But I use Windows/Linux/MacOS/…? - works on all platforms!
- But how about Windows CMD or Power-Shell? - works!
- But what if I use cygwin or git-bash on windows? - works!
- But I love to use ConEmu or Commander? - works with full integration!
- How about MacOS Terminal or iTerm2? - works with full integration!
- But I use zsh? - works!
- …? - works!

Wow! So lets get started with download & setup.

2. Setup

2.1. Prerequisites

We try to make it as simple as possible for you. However, there are some minimal prerequisites:

- You need to have a tool to extract `*.tar.gz` files (`tar` and `gzip`). On Windows use 7zip. On all other platforms this comes out of the box.
- You need to have git and curl installed.
  - On Windows all you need is download and install git for windows. This also ships with `bash` and `curl`.
  - On Linux you might need to install the above tools in case they are not present (e.g. `sudo apt-get install git curl` or `yum install git-core curl`)
  - On MacOS all you need is download and install git for mac.

2.2. ATTENTION

The official public release of `devonfw-ide` is still pending due to some final clarification. The download links are not broken, but pointing to the place where the release will appear after the clarification is complete. If you want to use `devonfw-ide` today you can clone it from ide.git and run `mvn install` to build it, or contact us (check internal community annoucements).

2.3. Download

Releases of `devonfw-ide` will be published to maven central.

2.4. Install

Create a central folder like `C:\projects` or `/projects`. Inside this folder create a sub-folder for your new project such as `my-project` and extract the contents of the downloaded archive (`devonfw-ide-scripts-*.tar.gz`) into this new folder. Run the command `setup` in this folder (on windows just double click `setup.bat`). That's all. To get started read the usage.

2.5. Uninstall

To "uninstall" your `devonfw-ide` you only need to call the following command:

```
devon ide uninstall
```

Then you can delete the `devonfw-ide` top-level folder(s) (${DEVON_IDE_HOME}).

The `devonfw-ide` is designed to be **not invasive** to your operating system and computer. Therefore it is not "installed" into your system in a classical way. Instead you just create a folder and extract the downloaded archive to it. Only specific prerequisites like `git` have to be installed regularly by you in advance. All other software resists just locally inside the folder of your `devonfw-ide`. However, there are the following excuses (what is reverted by `devon ide uninstall`):

- The `devon` command is copied to your home directory (`~/.devon/devon`)
- The `devon` alias is added to your shell config (`~/.bashrc` and `~/.zshrc`, search for `alias devon="source ~/.devon/devon"`).
- On Windows the `devon.bat` command is copied to your home directory (`%USERPROFILE%\scripts\devon.bat`)
- On Windows this `%USERPROFILE%\scripts` directory is added to the `PATH` of your user.
- The `devonfw-ide` will download third party software to your `~/Downloads` folder to reduce redundant storage.

3. Usage

This section explains you how to use the `devonfw-ide`. We assume you have successfully installed it first.

3.1. Developer

As a developer you are supported to setup your IDE within minutes. You only need the settings URL from your Architect.

3.1.1. Update

To update your IDE (if instructed by your Architect), all you need to do is run the following command:

```
devon ide update
```

3.1.2. Working with multiple workspaces

If you are working on different branches in parallel you typically want to use multiple workspaces.

1. Go to the workspaces folder in your ${DEVON_IDE_HOME} and create a new folder with the name of your choice (e.g. `release2.1`).
2. Check out (`git clone …`) the according projects and branch into that workspace folder.
3. Open a shell in that new workspace folder (`cd` to it) and according to your IDE run e.g. eclipse, vscode, or intellij to create your workspace and launch the IDE. You can also add the parameter `create-script` to the IDE commandlet in order to create a launch-script for your IDE.

You can have multiple instances of eclipse running for each workspace in parallel. To distinguish these instances you will find the workspace name in the title of eclipse.

3.2. Architect

As architect or technical lead of the project you can configure the `devonfw-ide` to your needs.

3.2.1. Project specific settings

For your project you should create a git-repository for the settings. You can customize many aspects this way.

3.2.2. Distribute

To redistribute the IDE you can decide to use the official vanilla releases of the `devonfw-ide` scripts. However, you may also add the cloned settings, a custom devon.properties file, or predefine software (be aware of multi-platform-support).

3.2.3. Update

When you have done changes in a larger project (big team), please first test the changes yourself, then pick a pilot user of the team, and only after that works well for a couple of days inform the entire team to update.

4. Configuration

The `devonfw-ide` aims to be highly configurable and flexible. The configuration of the devon command and environment variables takes place via `devon.properties` files. The following list shows these configuration files in the order they are loaded so files can override variables from files above in the list:

1. build in defaults (for `JAVA_VERSION`, `ECLIPSE_PLUGINS`, etc.)
2. `~/devon.properties` - user specific global defaults (on windows in `%USERPROFILE%/devon.properties`)
3. `scripts/devon.properties` - defaults provided by `devonfw-ide`. Never directly modify this file!
4. `devon.properties` - vendor variables for custom distributions of `devonfw-ide-scripts`, may e.g. tweak `SETTINGS_PATH` or predefine `SETTINGS_URL`.
5. `settings/devon.properties` (`${SETTINGS_PATH}/devon.properties`) - project specific configurations from settings.
6. `workspaces/${WORKSPACE}/devon.properties` - optional workspace specific configurations (especially helpful in projects using docker).
7. `conf/devon.properties` - user specific configurations (e.g. `M2_REPO=~/.m2/repository`). During setup this file is created by copying a template from `${SETTINGS_PATH}/devon/conf/devon.properties`.

4.1. devon.properties

The `devon.properties` files allow to define environment variables in a simple and OS independent way:

- `# comments begin with a hash sign (#) and are ignored`
- `variable_name=variable_value with space etc.`
- `variable_name=${predefined_variable}/folder_name`

  variable values can refer to other variables that are already defined what will be resolved to their value. You have to used `${…}` syntax to make it work on all platforms (never use `%…%`, `$…`, or `$(…)` syntax in `devon.properties` files).

- `export exported_variable=this value will be exported in bash, in windows CMD the export prefix is ignored`
- `variable_name=`

  this will unset the specified variable

- `variable_name=~/some/path/and.file`

  tilde is resolved to your personal home directory on any OS including windows.

- `array_variable=(value1 value2 value3)`

  This will only work properly in `bash` worlds but as no arrays are used in CMD world of `devonfw-`

`ide` it does not hurt on windows.

- Please never surround values with quotes (`var="value"`)
- This format is similar to Java `*.properties` but does not support advanced features as unicode literals, multi-lined values, etc.

In order to know what to configure have a look at the available variables.

Please only tweak configurations that you need to change and take according responsibility. Flexibility comes with some price. Of course you can break everything if you do the wrong things.

Further, you can configure maven via `conf/settings.xml`. To configure your IDE such as eclipse or vscode you can tweak the settings.

5. Variables

The devonfw-ide defines a set of standard variables to your environment for configuration via variables[.bat] files. These environment variables are described by the following table. Those variables printed **bold** are also exported in your shell (except for windows CMD that does not have such concept). Variables with the value - are not set by default but may be set via configuration to override defaults. Please note that we are trying to minimize any potential side-effect from devonfw-ide to the outside world by reducing the number of variables and only exporting those that are required.

Table 1. Variables of devonfw-ide

Variable	Value	Meaning
DEVON_IDE_HOME	e.g. /projects/my-project	The top level directory of your devonfw-ide structure.
PATH	$PATH:$DEVON_IDE_HOME/software/java:…	You system path is adjusted by devon command.
DEVON_HOME_DIR	~	The platform independent home directory of the current user. In some edge-cases (e.g. in cygwin) this differs from ~ to ensure a central home directory for the user on a single machine in any context or environment.
DEVON_IDE_TOOLS	java mvn eclipse vscode node npm ng	List of tools that should be installed and upgraded by default for your current IDE.
DEVON_IDE_CUSTOM_TOOLS	-	List of custom tools that should be installed additionally. See software for further details.
DEVON_OLD_PATH	…	A "backup" of PATH before it was extended by devon to allow recovering it. Internal variable that should never be set or tweaked.
WORKSPACE	main	The workspace you are currently in. Defaults to main if you are not inside a workspace. Never touch this variable in any varibales file.
WORKSPACE_PATH	$DEVON_IDE_HOME/workspaces/$WORKSPACE	Absolute path to current workspace. Never touch this variable in any varibales file.
JAVA_HOME	$DEVON_IDE_HOME/software/java	Path to JDK

Variable	Value	Meaning
`SETTINGS_PATH`	`$DEVON_IDE_HOME/settings`	Path to your settings. To keep `oasp4j-ide` legacy behaviour set this to `$DEVON_IDE_HOME/workspaces/main/development/settings`.
`M2_REPO`	`$DEVON_IDE_HOME/conf/.m2/repository`	Path to your local maven repository. For projects without high security demands, you may change this to the maven default `~/.m2/repository` and share your repository amongst multiple projects.
`MAVEN_HOME`	`$DEVON_IDE_HOME/software/maven`	Path to Maven
`MAVEN_OPTS`	`-Xmx512m -Duser.home=$DEVON_IDE_HOME/conf`	Maven options
`DEVON_SOFTWARE_REPOSITORY`	`-`	Project specific or custom software-repository.
`DEVON_SOFTWARE_PATH`	`-`	Globally shared user-specific local software installation location.
`ECLIPSE_VMARGS`	`-Xms128M -Xmx768M -XX:MaxPermSize=256M`	JVM options for Eclipse
`ECLIPSE_PLUGINS`	`-`	Array with "feature groups" and "update site URLs" to customize required eclipse plugins.
`«TOOL»_VERSION`	`-`	The version of the tool `«TOOL»` to install and use (e.g. `ECLIPSE_VERSION` or `MAVEN_VERSION`).
`«TOOL»_BUILD_OPTS`	e.g.`clean install`	The arguments provided to the build-tool `«TOOL»` in order to run a build.
`«TOOL»_RELEASE_OPTS`	e.g.`clean deploy -Dchangelist= -Pdeploy`	The arguments provided to the build-tool `«TOOL»` in order to perform a release build.

6. Devon CLI

The `devonfw-ide` is shipped with a central command `devon`. The setup will automatically register this command so it is available in any shell on your system. This page describes the Command Line Interface (CLI) of this command.

6.1. Devon

Without any arguments the `devon` command will determine your `DEVON_IDE_HOME` and setup your environment variables automatically. In case you are not inside of a `devonfw-ide` folder the command will echo a message and do nothing.

```
[/]$ devon
You are not inside a devon IDE installation: /
[/]$ cd /projects/my-project/workspaces/test/my-git-repo
[my-git-repo]$ devon
devonfw-ide has environment variables have been set for /projects/my-project in
workspace main
[my-git-repo]$ echo $DEVON_IDE_HOME
/projects/devon
[my-git-repo]$ echo $JAVA_HOME
/projects/my-project/software/java
```

6.2. Commandlets

The `devon` command supports a pluggable set of *commandlets*. Such commandlet is provided as first argument to the devon command and my take additional arguments:

`devon «commandlet» [«arg»]*`

Technically a commandlet is a bash script located in `$DEVON_IDE_HOME/scripts/command`. So if you want to integrate another tool with `devonfw-ide` we are awaiting your pull-request.

The following commandlets are currently available:

- build
- eclipse
- gradle
- help
- ide
- intellij
- java
- jenkins
- mvn

- ng
- node
- npm
- release
- sonar
- vscode
- yarn

6.3. build

The `build` commandlet is an abstraction of build systems like maven, gradle, yarn, npm, etc. It will auto-detect your build-system (via existence of files like `pom.xml`, `package.json`, etc.). According to this detection it will simply delegate to the according commandlet of the specific build system. If that build-system is not yet available it will be downloaded and installed automatically.

So `devon build` allows users to build any project without bothering about the build-system. Further specific build options can be configured per projects what allows `devon build` to be a universal part of every *definition of done*. Before pushing your changes simply always run the following command to verify the build:

```
devon build
```

You may also supply additional arguments as `devon build «args»`. This will simply delegate these arguments to the detected build command (e.g. call `mvn «args»`).

6.4. eclipse

The `eclipse` commandlet allows to install, configure, and launch the Eclipse IDE. To launch Eclipse for your current workspace and devonfw-ide installation simply run: `devon eclipse`

You may also supply additional arguments as `devon eclipse «args»`. These are explained by the following table:

Table 2. Usage of `devon eclipse`

Argument(s)	Meaning
`--all`	if provided as first arg then to command will be invoked for each workspace
`setup`	setup Eclipse (install or update)
`add-plugin «id» [«url»]`	install an additional plugin
`run`	launch Eclipse (default if no argument is given)
`start`	same as `run`
`ws-up[date]`	update workspace
`ws-re[verse]`	reverse merge changes from workspace into settings

Argument(s)	Meaning
`ws-reverse-add`	reverse merge adding new properties
`create-script`	create launch script for this IDE, your current workspace and your OS

6.4.1. plugins

During the setup `devon eclipse setup` will be called what automatically installs Eclipse. The project configuration typically defines the plugins that will be installed via `ECLIPSE_PLUGINS` variable. Otherwise defaults from this `eclipse` commandlet will apply. However, for specific needs you may even install additional plugins. In general you should try to stick with the configuration pre-defined by your project. But some plugins may be considered as personal flavor and are typically not predefined by the project config. This e.g. applies for devstyle that allows a real dark mode for eclipse and tunes the theming and layout of Eclipse in general. To avoid that projects ship with too many plugins that may waste resources but are not used by every developer you have the freedom to install some additional plugins. Be aware that this comes at your own risk and sometimes plugins can conflict and break your IDE. The following list of plugins is guaranteed to be supported by this commandlet:

- cobigen (incremental code-generator)
- anyedit (for easy compare with clipboard, etc.)
- checkstyle (for checkstyle support)
- spotbugs (successor of findbugs)
- startexplorer (for support to open current item in file manager of your OS)
- terminal (open terminal/shell inside Eclipse as view)
- github (for devonfw projects that want to access github issues in Eclipse)
- soapui (for service testing)
- regexutil (to test regular expressions)
- templatevariables (for advanced JDT templates)

The link-titles are the IDs accepted by `devon eclipse add-plugin «id»`. Otherwise the full featureID has to be specified together with the URL of the update site. However this is intended for project specific configuration. Here is an example how a project can configure the plugins in its `devon.properties` inside the settings:

```
ECLIPSE_PLUGINS=("AnyEditTools.feature.group" "http://andrei.gmxhome.de/eclipse/"
"com.ess.regexutil.feature.group" "http://regex-util.sourceforge.net/update/")
```

For the above listed plugins you can also use the short form:

```
ECLIPSE_PLUGINS=("anyedit" "" "regexutil" "")
```

Of course you may also mix plugin IDs with fully qualified plugins.

6.5. gradle

The `gradle` commandlet allows to install, configure, and launch gradle. It is similar to gradle-wrapper. So calling `devon gradle «args»` is more or less the same as calling `gradle «args»` but with the benefit that the version of gradle preferred by your project is used (and will be installed if not yet available).

The arguments (`devon gradle «args»`) are explained by the following table:

Table 3. Usage of `devon gradle`

Argument(s)	Meaning
`setup`	setup gradle (install and verify), configurable via `GRADLE_VERSION`
`«args»`	run gradle with the given arguments (`«args»`)

6.6. help

The `help` commandlet provides help for the CLI.

Table 4. Usage of `devon help`

Argument(s)	Meaning
	Print general help
`«command»`	Print help for the commandlet `«command»`.

Please note that `devon help «command»` will do the same as `devon «command» help`.

6.7. ide

The `ide` commandlet manages your `devonfw-ide`. You need to supply additional arguments as `devon ide «args»`. These are explained by the following table:

Table 5. Usage of `devon ide`

Argument(s)	Meaning
`setup [«SETTINGS_URL»]`	setup devonfw-ide (cloning the settings from the given URL)
`update [«package»]`	update devonfw-ide
`update scripts [to «version»]`	update devonfw-ide
`uninstall`	uninstall devonfw-ide (if you want remote it entirely from your system)

6.7.1. setup

Run `devon ide setup` to initially setup your `devonfw-ide`. It is recommended to run the `setup` script in the toplevel directory (`$DEVON_IDE_HOME`). However, in case you want to skip some system specific integrations, you may also run this command directly instead. The setup only needs to be called once after a new `devonfw-ide` instance has been created. It will do the following things:

- install the `devon` command on your system (if not already installed).
- clone the settings (you may provide a git URL directly as argument or you will be prompted for it).
- install all required software from `DEVON_IDE_TOOLS` variable (if not already installed).
- configure all these tools
- create IDE launch scripts
- perform OS specific system integration such as WindowsExplorer integration (only done from `setup` script and not from `devon ide setup`)

6.7.2. update

Run `devon ide update` to update your `devonfw-ide`. This will check for updates and install them automatically. The optinal extra argument (`«package»`) behaves as following:

- `scripts`: check if a new version of `devonfw-ide-scripts` is available. If so it will be downloaded and installed. As Windows is using file-locks, it is tricky to update a script while it is executed. Therefore, we update the `scripts` folder as an async background task and have to abort further processing at this point on windows as a workaround.
- `settings`: update the settings (`git pull`).
- `software`: update the software (e.g. if versions have changed via `scripts` or `settings` update).
- `all`: do all the above sequentially.
- none: `settings` and `software` is updated by default if no extra argument is given. This is the regular usage for project developers. Only perform an update of `scripts` when you are requested to do so by your technical lead. Especially bigger projects need to test updates before rolling them out to the entire team. If developers would always update to the latest release of the `scripts` which is released globally, some project functionality might break causing problems and extra efforts in the teams.

In order to update to a specific version of `scripts` an explicit version can be specified after the additional `to` argument:

```
devon ide update scripts to 3.1.99
```

The above example will update to the exact version `3.1.99` no matter if this is an upgrade or a downgrade of your current installed version. If you just use `devon ide update scripts` then the latest available version will be installed. In larger teams it is recommended to communicate exact version updates to avoid that a new release can interfere and break anything. Therefore some pilot user will test a new version for the entire team and only after successful test will communicate to the team to update to that exact version by providing the complete command as in the above example.

6.7.3. uninstall

We hope you love `devonfw-ide`. However, if you don't and want to get rid of it entirely and completely remove all integrations, you can use this command:

```
devon ide uninstall
```

This will remove `devonfw-ide` from all central places of your OS (user home directory such as `scripts`, `.devon`, `.bashrc`, as well as windows registry, etc.). However, it will not remove your actual installations (or shared software folder). So after running this `uninstall`, simply remove your `DEVON_IDE_HOME` directory of all `devonfw-ide` installations and potential shared software folder. You may also want to clean up your `~/Downloads` directory from files downloaded by `devonfw-ide`. We do not automate this as deleting a directory is a very simple manual step and we do not want to take responsibility for severe data loss if your workspaces contained valuable work.

6.8. vscode

The `intellij` commandlet allows to install, configure, and launch IntelliJ. To launch IntelliJ for your current workspace and `devonfw-ide` installation simply run: `devon intellij`

You may also supply additional arguments as `devon intellij «args»`. These are explained by the following table:

Table 6. Usage of `devon intellij`

Argument(s)	Meaning
`--all`	if provided as first arg then to command will be invoked for each workspace
`setup`	setup IntelliJ (install or update)
`run`	launch IntelliJ (default if no argument is given)
`start`	same as `run`
`ws-up[date]`	update workspace
`ws-re[verse]`	reverse merge changes from workspace into settings
`ws-reverse-add`	reverse merge adding new properties
`create-script`	create launch script for this IDE, your current workspace and your OS

6.9. java

The `java` commandlet allows to install and setup Java. Also it supports devon4j. The arguments (`devon java «args»`) are explained by the following table:

Table 7. Usage of `devon java`

Argument(s)	Meaning
`setup`	setup OpenJDK (install or update and verify), configurable via `JAVA_VERSION` (e.g. `8u222b10` or `11.0.4_11`)
`create «args»`	create a new Java project based on devon4j application template. If a single argument is provided, this is the package name and is automatically split into groupId and artifactId. Use `-DdbType=«db»` to choose the database (hana, oracle, mssql, postgresql, mariadb, mysql, h2, hsqldb). Any option starting with dash is passed as is."
`migrate [from «version»] [single]`	migrate a devon4j project to the latest version. If for some reasons the current devonfw version (e.g. oasp4j:2.6.0) can not be auto-detected you may provide it manually after the 'from' argument. Also the 'single' option allows to migrate only to the next available version."
`cicd «args»`	generate cicd files for the currect devon4java project

6.9.1. create

Examples for create a new devon4j application:

```
devon java create com.example.domain.myapp
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.domain`, artifactId `myapp`, version `1.0.0-SNAPSHOT`, and h2 database.

```
devon java create -Dversion=0.0.1-alpha1 com.example.domain.myapp
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.domain`, artifactId `myapp`, version `0.0.1-alpha1`, and h2 database.

```
devon java create com.example.domain.myapp com.example.group
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `myapp`, version `1.0.0-SNAPSHOT`, and h2 database.

```
devon java create com.example.domain.myapp com.example.group demo-app
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `demo-app`, version `1.0.0-SNAPSHOT`, and h2 database.

```
devon java create com.example.domain.myapp -DartifactId=demo-app -DdbType=hana
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `demo-app`, version `1.0.0-SNAPSHOT`, and SAP hana database.

```
devon java create com.example.domain.myapp -DdbType=oracle -Dversion=0.0.1
com.example.group -Dbatch=batch
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `myapp`, version `0.0.1`, oracle database, and with a batch module.

6.10. jenkins

The `jenkins` commandlet allows to install, configure, and launch Jenkins.

Table 8. Usage of `devon jenkins`

Argument(s)	Meaning
`setup`	Setup Jenkins (install and verify)
`start`	Start your local Jenkins server
`stop`	Stop your local Jenkins server
`add`	Add current project as CI job to your local Jenkins

6.11. mvn

The `mvn` commandlet allows to install, configure, and launch maven. It is similar to maven-wrapper and mdub. So calling `devon mvn «args»` is more or less the same as calling `mvn «args»` but with the benefit that the version of maven preferred by your project is used (and will be installed if not yet available).

The arguments (`devon mvn «args»`) are explained by the following table:

Table 9. Usage of `devon mvn`

Argument(s)	Meaning
	run default build, configurable via `MVN_BUILD_OPTS`
`setup`	setup Maven (install and verify), configurable via `MAVEN_VERSION`

Argument(s)	Meaning
`get-version`	Print the version of your current project. Will consolidate the version for multi-module projects ignoring `dev[-SNAPSHOT]` versions and fail on mixed versions.
`set-version` «nv» [«cv»]	Set the version of your current project to `«nv»` (assuming your current version is `«cv»`).
`check-top-level-project`	Check if you are running on a top-level project or fail if in a module or no maven project at all.
release	Start a clean deploy release build, configurable via `MVN_RELEASE_OPTS`
`«args»`	run maven with the given arguments (`«args»`)

6.12. ng

The `ng` commandlet allows to install, configure, and launch ng (angular-cli). Calling `devon ng «args»` is more or less the same as calling `ng «args»` but with some advanced features and ensuring that `ng` is properly set up for your project.

The arguments (`devon ng «args»`) are explained by the following table:

Table 10. Usage of `devon ng`

Argument(s)	Meaning
`setup`	setup yarn (install and verify), configurable via `YARN_VERSION`
`create`	Create a new devon4ng project.
`cicd «args»`	generate cicd files for the currect devon4ng project
`«args»`	run ng with the given arguments (`«args»`)

6.13. node

The `node` commandlet allows to install and setup node.js. The arguments (`devon node «args»`) are explained by the following table:

Table 11. Usage of `devon node`

Argument(s)	Meaning
`setup`	setup node.js (install and verify), configurable via `NODE_VERSION`
`create «name» [«args»]`	create a new devon4node application `(same as devon4node new)`
`generate «s» [«args»]`	generate devon4node components using the schematic «s» `(same as devon4node generate)`

Argument(s)	Meaning
`db «c» [«args»]`	execute a TypeORM command «c» `(same as devon4node db)`
`cicd «args»`	generate cicd files for the currect devon4node project
`«args»`	call NodeJS with the specified arguments

6.14. npm

The `npm` commandlet allows to install, configure, and launch npm. Calling `devon npm «args»` is more or less the same as calling `npm «args»` but with the benefit that the version of npm preferred by your project is used (and will be installed if not yet available).

The arguments (`devon npm «args»`) are explained by the following table:

Table 12. Usage of `devon npm`

Argument(s)	Meaning
	run default build, configurable via `NPM_BUILD_OPTS`
`setup`	setup NPM (install and verify), configurable via `NPM_VERSION`
`get-version`	print the version of your current project
`set-version «nv» [«cv»]`	set the version of your current project to `«nv»` (assuming your current version is `«cv»`)
`check-top-level-project`	check if you are running on a top-level project or fail if in a module or no NPM project at all
`release`	Start a clean deploy release build, configurable via `NPM_RELEASE_OPTS`
`«args»`	run NPM with the given arguments (`«args»`)

6.15. release

Create a release in a standardized way including the following steps:

- verify the current project (no local changes, etc.)
- determine `«version»` (if currently `«version»-SNAPSHOT`) and print out release information.
- ask user for confirmation
- bump release to `«version»` in build configuration (e.g. `pom.xml` files)
- commit the change
- create annotated tag for your release as `release/«version»`
- invoke deployment on build-system
- set next version as `(«version»+1)-SNAPSHOT` in build configuration (e.g. `pom.xml` files)

- commit the change
- push your changes

6.16. sonar

The `sonar` commandlet allows to install, configure, and launch SonarQube.

Table 13. Usage of `devon sonar`

Argument(s)	Meaning
`setup`	Setup SonarQube (install and verify)
`start`	Start your local SonarQube server
`stop`	Stop your local SonarQube server
`analyze`	Analyze current project with SonarQube

6.17. vscode

The `vscode` commandlet allows to install, configure, and launch Visual Studio Code. To launch VSCode for your current workspace and `devonfw-ide` installation simply run: `devon vscode`

You may also supply additional arguments as `devon vscode «args»`. These are explained by the following table:

Table 14. Usage of `devon vscode`

Argument(s)	Meaning
`--all`	if provided as first arg then to command will be invoked for each workspace
`setup`	setup VSCode (install or update)
`run`	launch VSCode (default if no argument is given)
`start`	same as `run`
`ws-up[date]`	update workspace
`ws-re[verse]`	reverse merge changes from workspace into settings
`ws-reverse-add`	reverse merge adding new properties
`create-script`	create launch script for this IDE, your current workspace and your OS

6.18. yarn

The `yarn` commandlet allows to install, configure, and launch npm. Calling `devon yarn «args»` is more or less the same as calling `yarn «args»` but with the benefit that the version of npm preferred by your project is used (and will be installed if not yet available).

The arguments (`devon yarn «args»`) are explained by the following table:

Table 15. Usage of `devon yarn`

Argument(s)	Meaning
	run default build, configurable via `YARN_BUILD_OPTS`
`setup`	setup yarn (install and verify), configurable via `YARN_VERSION`
`get-version`	print the version of your current project
`set-version «nv» [«cv»]`	set the version of your current project to `«nv»` (assuming your current version is `«cv»`)
`check-top-level-project`	check if you are running on a top-level project or fail if in a module or no NPM project at all
`release`	start a clean deploy release build, configurable via `YARN_RELEASE_OPTS`
`«args»`	run yarn with the given arguments (`«args»`)

7. Structure

The directory layout of your devonfw-ide will look like this:

File structure of your devonfw-ide

```
/ projects (or C:\Projects, etc.)
└──/ my-project ($DEVON_IDE_HOME)
    ├──/ conf
    ├──/ log
    ├──/ scripts
    ├──/ settings
    ├──/ software
    ├──/ system
    ├──/ updates
    ├──/ workspaces
    ├── setup
    ├── setup.bat
    └── TERMS_OF_USE.adoc
```

The elements of the above structure are described in the individual sections. As they are hyperlinks you can simply click them to get more details.

7.1. TERMS_OF_USE

You need to agree with the TERMS_OF_USE in order to use devonfw-ide. Everything included out of the box applies to open-source licenses. However, due to the many third party components with their licenses and terms we want to make this clear to all our users and be compliant.

7.2. conf

This folder contains configurations for your IDE:

File structure of the conf folder

```
/ conf
├──/ .m2
|   ├──/ repository
|   |   ├──/ ant
|   |   ├──/ ...
|   |   └──/ zw
|   ├── settings-security.xml
|   └── settings.xml
├──/ .sonar
├──/ ...
└── variables
```

The .m2 folder is used for configurations of maven. It contains the local repository folder used as cache for artifacts downloaded and installed by maven (see also maven repositories). Further there are two configuration files for maven:

- settings.xml initialized from a template from your `devonfw-ide` settings. You may customize this to your needs (configuring HTTP proxies, credentials, or other user-specific settings).
- settings-security.xml is auto-generated for you by `devonfw-ide` with a random password. This should make it easier for `devonfw-ide` users to use password encryption and never add passwords in plain text for better security.

Finally there is a file `variables` for the user-specific configuration of `devonfw-ide`.

7.3. log

The log directory is used to store log files e.g. for the IDE configurator. You may look here for debug information if something goes wrong.

7.4. scripts

This directory is the heart of the `devonfw-ide` and contains the required scripts.

File structure of the conf folder

```
/scripts
├──/ command
│   ├── build
│   ├── eclipse
│   ├── gradle
│   ├── help
│   ├── ide
│   ├── intellij
│   ├── java
│   ├── jenkins
│   ├── mvn
│   ├── ng
│   ├── node
│   ├── npm
│   ├── release
│   ├── sonar
│   ├── vscode
│   └── yarn
├── devon
├── devon.bat
├── environment-project
├── environment-project.bat
├── functions
└── devon.properties
```

The command folder contains the commandlets. The devon script is the key command line interface for `devonfw-ide`. For windows there is also devon.bat to be used in CMD or PowerShell. As the devon CLI can be used as global command on your computer from any directory and gets installed centrally it aims to be stable, minimal, and lightweight. The key logic to setup the environment variables is therefore in a separate script environment-project and its Windows variant environment-project.bat inside this scripts folder. The file functions contains a collection of reusable bash functions. These are sourced and used by the commandlets. Finally the devon.properties file contains defaults for the general configuration of devonfw-ide.

7.5. settings

The devonfw-ide requires settings with configuration templates for the arbitrary tools.

To get an initial set of these settings we provide the devonfw-ide-settings as an initial package. These are also released so you can download the latest stable version at maven central.

To test devonfw-ide or for small projects you can also use these the latest default settings. However, for collaborative projects we strongly encourage you to distribute and maintain the settings via a dedicated and project specific git repository (or any other version-control-system). This gives you the freedom to control and manage the tools with their versions and configurations during the project lifecycle. Therefore a technical lead creates a settings git repository for the project. He creates a "fork" devonfw-ide-settings by adding it as "upstream" origin and pulls from there finally pushing it to the project settings git. This also allows to later merge changes from the official devonfw-ide-settings back into the project specific settings "fork".

7.5.1. Structure

The settings folder (see SETTINGS_PATH) has to following file structure:

File structure of settings

```
/settings
 ├──── / devon
 |     ├──── / conf
 |     |     ├──── / .m2
 |     |     |     └──── settings.xml
 |     |     ├──── / npm
 |     |     |     └──── .npmrc
 |     |     └──── devon.properties
 ├──── / eclipse
 |     ├──── / workspace
 |     |     ├──── / setup
 |     |     └──── / update
 |     ├──── lifecycle-mapping-metadata.xml
 |     └──── project.dictionary
 ├──── / ...
 ├──── / sonarqube
 |     └──── / profiles
 |        ├──── Devon-C#.xml
 |        ├──── ...
 |        └──── Devon-XML.xml
 ├──── / vscode
 |     └──── / workspace
 |        ├──── / setup
 |        └──── / update
 └──── devon.properties
```

As you can see the `settings` folder contains sub-folders for tools of the IDE. So the `devon` folder contains `devon.properties` files for the configuration of your environment. Further, for the IDEs such as eclipse or vscode the according folders contain the templates to manage the workspace via our configurator.

7.5.2. Configuration Philosophy

Different tools and configuration files require a different handling:

- Where suitable we directly use these configurations from your `settings` (e.g. for `eclipse/lifecycle-mapping-metadata.xml`, or `eclipse/project.dictionary`).
- The `devon` folder in `settings` contains templates for configuration files. There are copied to the `devonfw-ide` installation during setup (if no such file already exists). This way the `settings` repository can provide reasonable defaults but allows the user to take over control and customize to his personal needs (e.g. `.m2/settings.xml`).
- Other configurations need to be imported manually. To avoid manual steps and simplify usage we try to automate as much as possible. This currently applies to `sonarqube` profiles but will be automated with sonar-devon4j-plugin in the future.
- For tools with complex configuration structures like eclipse, intellij, or vscode we provide a smart mechanism via our configurator.

7.5.3. Customize Settings

You can easily customize these settings for the requirements of your project. We suggest that one team member is responsible to ensure that everything stays consistent and works.

You may also create new sub-folders in `settings` and put individual things according to your needs. E.g. you could add scripts for greasemonkey or tampermonkey, as well as scripts for your database or whatever may be useful and worth to share in your team. However, to share and maintain knowledge we recommend to use a wiki.

7.6. software

The `software` folder contains the third party tools for your IDE such as maven, npm, java, etc. With respect to the licensing terms you may create a custom archive containing a `devonfw-ide` together with the required software. However, to be platform independent and allow lightweight updates the `devonfw-ide` is capable to download and install the software automatically for you.

7.6.1. Repository

By default software is downloaded via the internet from public download URLs of the according tools. However, some projects may need specific tools or tool versions that are not publicly available. In such case, they can create their own software repository (e.g. in a VPN) and configure the base URL of it via `DEVON_SOFTWARE_REPOSITORY` variable. Then `devonfw-ide` will download all software from this repository only instead of the default public download URLs. This repository (URL) should be accessible within your network via HTTPS (or HTTP) and without any authentication. The repository needs to have the following structure:

```
${DEVON_SOFTWARE_REPOSITORY}/«tool»/«version»/«tool»-«version»[-«os»].tgz
```

So for every tool `«tool»` (java, maven, vscode, eclipse, etc.) you need to provide a folder in your repository. Within this folder for every supported version `«version»` you need a subfolder. This subfolder needs to contain the tool in that version for every operating system `«os»` (`windows`, `linux`, or `mac` - omitted if platform independent, e.g. for `maven`).

7.6.2. Shared

By default each installation of `devonfw-ide` has its own physical installations of the required tools in the desired versions stored in its local `software` folder. While this is great for isolation of `devonfw-ide` installations and to prevent side-effects, it can cause a huge waste of disc resources in case you are having many installations of `devonfw-ide`. If you are a power-user of `devonfw-ide` with more then ten or even up to hundreds of installations on your machine, you might love to share installations of a software tool in a particular version between multiple `devonfw-ide` installations.

If you use this power-feature you are taking responsibility for side-effects and should not expect support. Also if you are using Windows please read Symlinks in Windows and make your mind if you really want to do so. You might also use this hint and maintain it manually without enabling the following feature.

In order to do so, you only need to configure the variable `DEVON_SOFTWARE_PATH` in your `~/devon.properties` pointing to an existing directory on your disc (e.g. `/projects/software` or `C:\projects\software`). Then `devonfw-ide` will install required software into `${DEVON_SOFTWARE_PATH}/${software_name}/${software_version}` as needed and create a symbolic link to it in `${DEVON_IDE_HOME}/software/${software_name}`.

As a benefit another `devonfw-ide` installation will using the same software with the same version can re-use the existing installation and only needs to create the symbolic link. No more waste of having many identical JDK installations on your disc.

As a drawback you need to be aware that specific tools may be "manipulated" after installation. The most common case is that a tool allows to install plugins or extensions such as all IDEs do. Such "manipulations" will cause side-effects between the different `devonfw-ide` installations sharing the same version of that tool. While this can also be a benefit it may also cause trouble. If you have a sensitive project that should not be affected by such side-effects, you may again override the `DEVON_SOFTWARE_PATH` variable to the empty value in your `${DEVON_IDE_HOME}/conf/devon.properties` of that sensitive installation:

```
DEVON_SOFTWARE_PATH=
```

This will disable this feature particularly for that specific sensitive `devonfw-ide` installation but let you use it for all other ones.

7.6.3. Custom

In some cases a project might need a (proprietary) tool(s) that is not supported by `devonfw-ide`. A very simple solution is to get a release of `devonfw-ide` and add the tool(s) to the software folder and then distribute this modified release to your team. However, this has several drawbacks as you then have a fork of `devonfw-ide` all will loose your tool(s) when updating to a new release.

As a solution for this need, `devonfw-ide` offers you to configure cutom tools via the `DEVON_IDE_CUSTOM_TOOLS` variable. It can be defined in `devon.properties` of your settings git repository as an array of the custom tools you need to add. Each entry applies:

- It needs to have the form `«tool»:«version»[:all][:«repository-url»]`
- The first entry must have the `«repository-url»` included which is used as default
- Further entries will inherit this default if omitted
- This URL is used in the same way as described above for a software repository.
- The `DEVON_SOFTWARE_REPOSITORY` variable is ignored by this feature.
- The optional infix `:all` is used to indicate that the tool is platform independent. Otherwise an OS specific infix is appended the the file URL to download for your platform (`windows`, `linux`, or `mac`).

As an example we define the following in `${DEVON_IDE_HOME}/settings/devon.properties`:

```
DEVON_IDE_CUSTOM_TOOLS=(jboss-eap:7.1.4.GA:all:https://host.tld/projects/my-project
firefox:70.0.1)
```

This will download and extract the following to your `software` folder:

- https://host.tld/projects/my-project/jboss-eap/7.1.4.GA/jboss-eap-7.1.4.GA.tgz
- https://host.tld/projects/my-project/firefox/70.0.1/firefox-70.0.1-windows.tgz

Please note that if you are not using windows, the `-windows` suffix will be `-mac` or `-linux`.

7.7. system

The system folder contains documentation and solutions for operation system specific integration. Please have a look to get the maximum out of `devonfw-ide` and become a very efficient power user.

7.8. updates

The `updates` folder is used for temporary data. This includes:

- extracted archives for installation and updates
- backups of old content on updates to prevent data loss

If all works fine you may clean this folder to save some kilo- or mega-bytes. Otherwise you can ignore it unless you are looking for a backup after a failed or unplanned upgrade.

7.9. workspaces

The `workspaces` folder contains folders for your active work. There is a workspace folder `main` dedicated for your primary work. You may do all your work inside the `main` workspace. Also you are free to create any number of additional workspace folders named as you like (e.g. `test`, `release`, `testing`, `my-sub-project`, etc.). Using multiple workspaces is especially relevant for Eclipse as each workspace has its own Eclipse runtime instance and configuration.

Within the workspace folder (e.g. `workspaces/main`) you are again free to create sub-folders for (sub-)projects according to your needs. We assume that in most cases you clone git repositories here. The following structure shows an example layout for devonfw:

File structure of workspaces

```
/ workspaces
 ├──── / main
 |    ├──── / .metadata
 |    ├──── / ide
 |    ├──── / devon4j
 |    └──── / my-thai-star
 └──── / stable
      ├──── / .metadata
      ├──── / ide
      └──── / devon4j
```

In the `main` workspace you may find the cloned forks for regular work (in the example e.g. `devon4j`) as a base to create pull-requests while in the `stable` workspace there is a clone of `devon4j` from the official devon4j. However this is just an example. Some people like to create separate workspaces for development and maintenance branches with git. Other people just switch between those via `git checkout`.

8. Advanced Tooling

8.1. Cross-Plattform Tooling

Git Client

If you are looking for a git client that works cross-platform we recommend to use Fork.

Browser Plugins

There are tons of helpful browser plugins out there and it might be a matter of personal taste what you like to have installed. However, as we are heavily using github we want to promote octotree. In case you also work with ZenHub you might want to install the Zenhub Browser Extension.

8.2. Windows Tooling

8.2.1. Integration into Windows-Explorer

After you have setup your `devonfw-ide` on a windows machine, you already have windows-explorer integration out-of-the-box. Just right-click on the folder you would like to open in a terminal and choose from the context menu:

- `Git Bash`
- `Open devonfw CMD shell here`
- `Open devonfw PowerShell here`
- `Open devonfw Cygwin Bash Here` (only if cygwin was installed during setup)

8.2.2. Tabs everywhere

Many people got used to *tabs* that have been introduced by all major browsers:

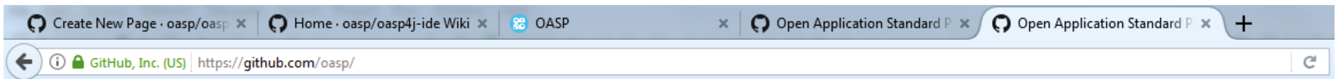

Create New Page · oasp/oasp
Home · oasp/oasp4j-ide Wiki
OASP
Open Application Standard P
Open Application Standard P
GitHub, Inc. (US)
https://github.com/oasp/

Figure 1. Tabs in Firefox

This nice feature can be added to many other tools.

Tabs for Windows Explorer

If you want to have tabs for windows explorer simply install Clover

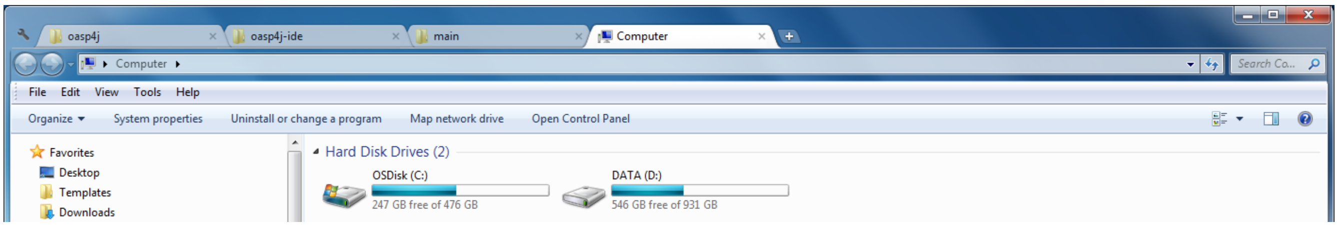
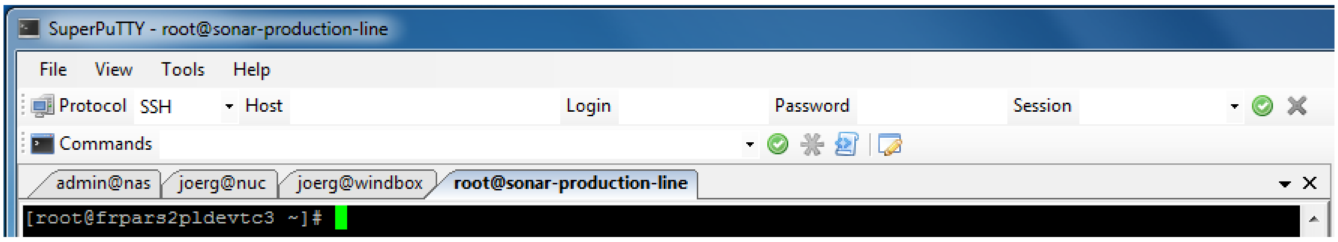


Figure 2. Tabs in Windows Explorer

Tabs for SSH

If you want to have tabs for your SSH client Putty (or even better Kitty that comes with WinSCP integration) you simply install SuperPutty

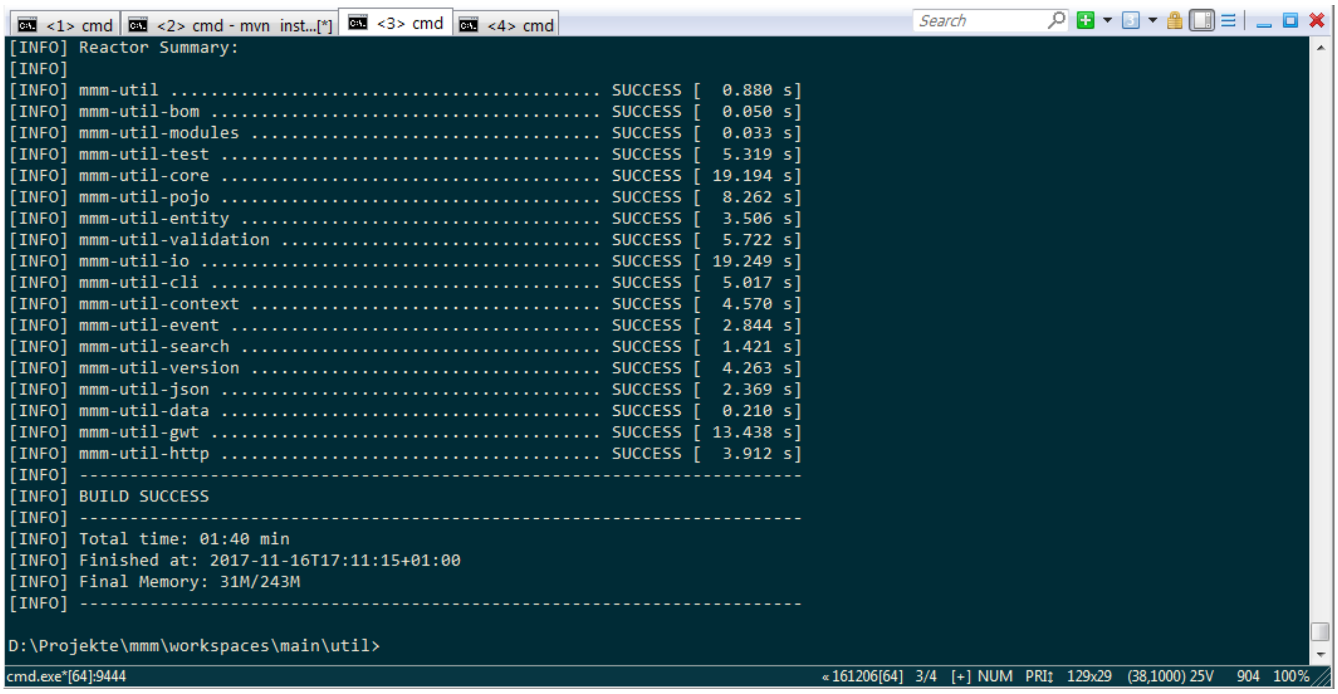

SuperPuTTY - root@sonar-production-line
File View Tools Help
Protocol SSH Host Login Password Session
Commands
admin@nas
joerg@nuc
joerg@windbox
root@sonar-production-line

```
[root@frpars2pldevtc3 ~]#
```

Figure 3. Tabs for SSH

Tabs for CMD

If you want to have tabs for your windows command-line you simply install ConEmu



```
[INFO] Reactor Summary:
[INFO]
[INFO] mmm-util ........................................... SUCCESS [  0.880 s]
[INFO] mmm-util-bom ....................................... SUCCESS [  0.050 s]
[INFO] mmm-util-modules ................................... SUCCESS [  0.033 s]
[INFO] mmm-util-test ...................................... SUCCESS [  5.319 s]
[INFO] mmm-util-core ...................................... SUCCESS [ 19.194 s]
[INFO] mmm-util-pojo ...................................... SUCCESS [  8.262 s]
[INFO] mmm-util-entity .................................... SUCCESS [  3.506 s]
[INFO] mmm-util-validation ................................ SUCCESS [  5.722 s]
[INFO] mmm-util-io ........................................ SUCCESS [ 19.249 s]
[INFO] mmm-util-cli ....................................... SUCCESS [  5.017 s]
[INFO] mmm-util-context ................................... SUCCESS [  4.570 s]
[INFO] mmm-util-event ..................................... SUCCESS [  2.844 s]
[INFO] mmm-util-search .................................... SUCCESS [  1.421 s]
[INFO] mmm-util-version ................................... SUCCESS [  4.263 s]
[INFO] mmm-util-json ...................................... SUCCESS [  2.369 s]
[INFO] mmm-util-data ...................................... SUCCESS [  0.210 s]
[INFO] mmm-util-gwt ....................................... SUCCESS [ 13.438 s]
[INFO] mmm-util-http ...................................... SUCCESS [  3.912 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:40 min
[INFO] Finished at: 2017-11-16T17:11:15+01:00
[INFO] Final Memory: 31M/243M
[INFO] ------------------------------------------------------------------------

D:\Projekte\mmm\workspaces\main\util>
cmd.exe*[64]:9444                      « 161206[64]  3/4  [+] NUM  PRI↕  129x29  (38,1000) 25V  904  100%
```

Figure 4. Tabs for CMD

See guide-integration.asciidoc for integration of ConEmu with `devonfw-ide`.

8.2.3. Windows Helpers

Handle passwords

For security you want complex passwords that differ for each account? For simplicity you only want to remember a single password? Want to have both? Then you need to install KeePass right now.

Real text editor

A real developer needs a real text editor and not windows build in `notepad`. The most common choice is Notepad++.

Real compression tool

You need to deal with ZIP files, TGZ, dpkg, etc.? Just install 7zip and forget about windows build-in ZIP support (that is buggy with long file paths, etc.).

Smarter clipboard

You want to paste something from the clipboard but meanwhile you had to copy something else? Just one of many things you can easily do with ditto.

Sysinternals Tools

A real developer will quickly notice that windows build in tools to analyze processes, network connections, autostarts, etc. are quite poor. So what you really want is the Sysinternals-Suite. Make process-explorer your default task manager asap. Use autoruns to prevent nasty background things to be started automatically. Use tcpview to figure out which process is blocking port 8080, etc.

Cope with file locks

Did you ever fail to delete a file or directory that was locked by some process and you did not even know which one it was? Then you might love IoBit Unlocker. See also this article.

Create symbolic links

You are used to symbolic and hard links in Linux? You have to work with Windows? You would also like to have such links in Windows? Why not? Windows supports real links (not these stupid shortcuts). If you even want to have it integrated in windows explorer you might want to install linkshellextension. However, you might want to disable `SmartMove` in the configuration if you face strange performance issues when moving folders.

Linux

Install Cygwin and get your bash in windows with ssh-agent, awk, sed, tar, and all the tools you love (or hate).

X11

Want to connect via SSH and need to open an X11 app from the server? Want to see the GUI on your windows desktop? No problem: Install VcXsrv.

Keyboard Freak

Are you the keyboard shortcut guy? Want to have shortcuts for things like « and » ? Then you should try AutoHotKey. For the example (« and ») you can simply use this script to get started:

```
^<::Send {U+00AB}
^+<::Send {U+00BB}
```

Now just press `[ctrl][<]` and `[ctrl][>]` (`[ctrl][shift][<]`). Next create shortcuts to launch your IDE, to open your favorite tool, etc.

Paint anywhere on your desktop

Do you collaborate sharing your screen, and want to mark a spot on top of what you see? Use Epic Pen to do just that.

analyze graphs

Need to visualise complex graph structures? Convert them to Trivial Graph Format (.tgf) an run yEd to get an interactive visualization of your graph.

up your screen capture game

Capture any part of your screen with a single click, directly upload to dropbox, or run an svn commit (oops sorry git ;-)) all in one go with Greenshot.

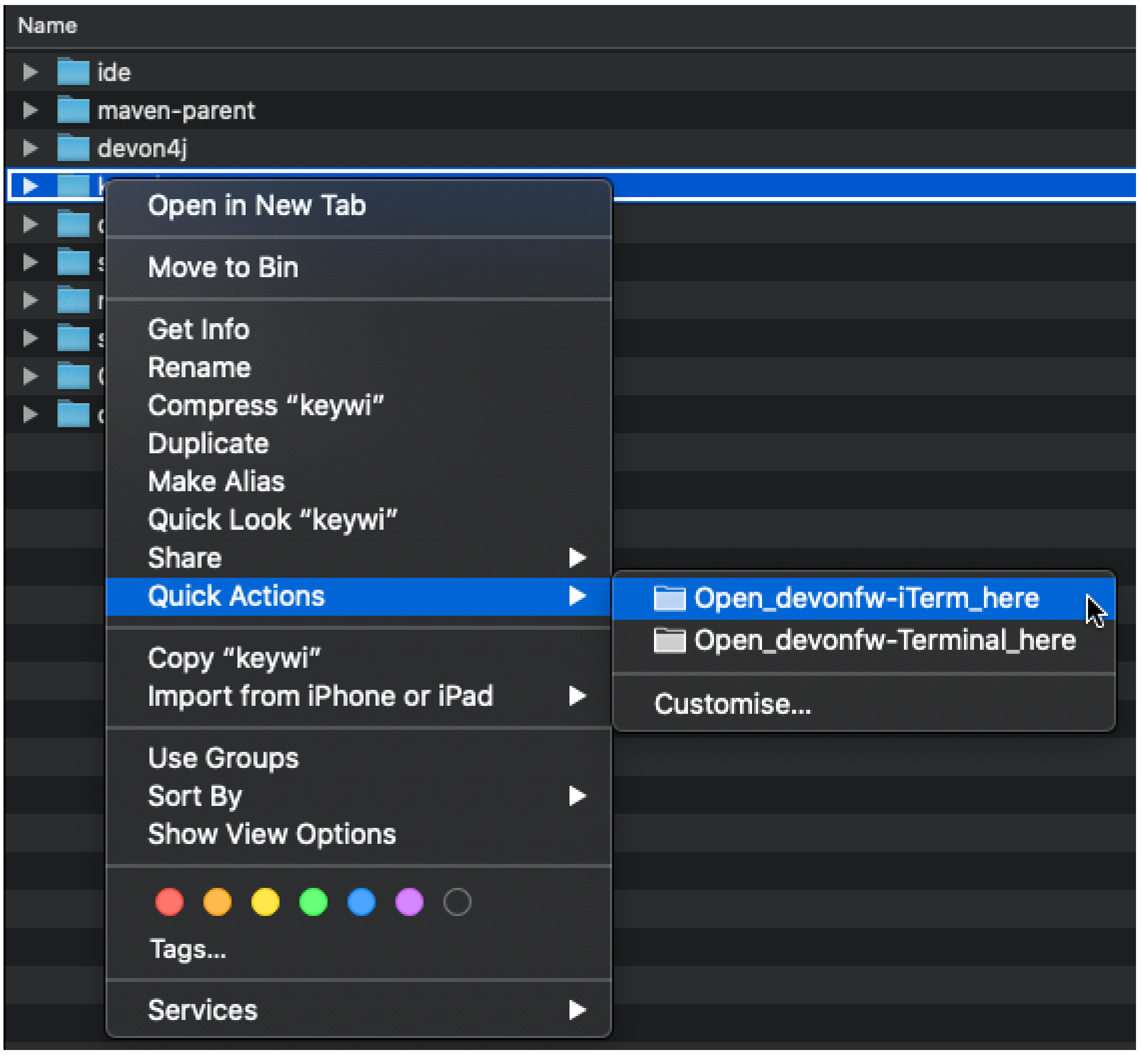
Fast Search in Windows

Everything is a desktop search utility for Windows that can rapidly find files and folders by name.

8.3. MacOS Tooling

8.3.1. Finder

If you want to open a terminal from a folder in `Finder` and automatically get your environment set properly for `devonfw-ide` you will find the perfect solution here.



So after installing (see below) the integration(s) provided here, you can easily open a terminal ready for your `devonfw-ide`:

- right click (`[control]` + click) on file or folder in `Finder`
- Expand the `Quick-Actions` sub-menu
- Click on the desired action (e.g. `Open devonfw-Terminal here`)
- Verify that you environment is properly initialized by invoking:

```
mvn -v
```

To get this feature for MacOS `Terminal.app` open `Finder` and run the workflow `system/mac/terminal/Open_devonfw-Terminal_here.workflow` (in `${DEVON_IDE_HOME}`). For `iTerm2.app` (can be installed from `App Store`) do the same with `system/mac/iterm/Open_devonfw-iTerm_here.workflow`.

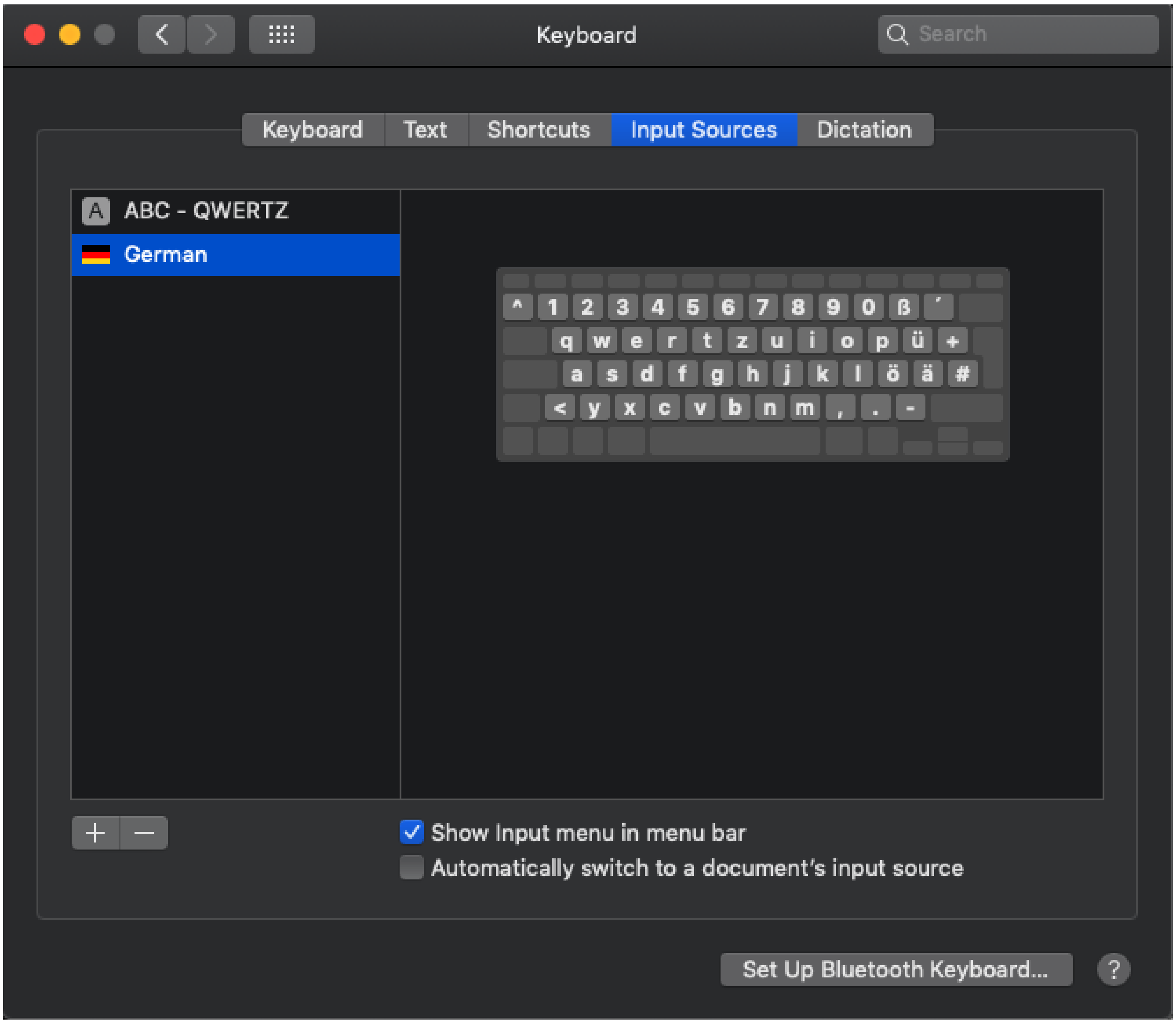
8.3.2. Keyboard

Keyboard support is not an integration but however, some users coming from other platforms may struggle with the way MacOS deals with (external non-apple) keyboards. So to make it short: if you are happy with your keyboard and shortcuts, you can skip all the following. Otherwise, if you think that pressing keys like `Home`, `End`, etc. should just work as expected or pressing `Alt Gr` should allow you to type the special characters as printed on your German keyboard then here you will find cure for your pain! To get all automated you can just run the script `system/mac/keyboard/install-mac-keyboard-support.sh` (in `${DEVON_IDE_HOME}`). In case you want to understand what is going on, you want to customizer the keyboard settings to your needs or you do not use German ISO keyboard layout, please read on.

Keyboard Layouts

Keyboard-Layouts allow a find grained mapping of each key on your keyboard to its resulting input character or behaviour. They are a MacOS native features and do not need to have software running as background service to make the keyboard mapping work (see Karabiner section below as alternative). They are provided as so called `bundle` (white lego brick icon). Like a MacOS app this is a folder containing a `Contents` folder with a specific sub-folder structure. In the `Resources` subfolder `*.keylayout` files are placed that define the exact mapping for the keyboard. As an example we provide a `Keyboard Layouts` folder containing a `bundle` for a German keyboard mapping.

To install keyboard layouts simply doubleclick the `bundle` or copy it to `~/Library/Keyboard Layouts`. To actually use them go to `System Preferences` and select `Keyboard`. Then select the tab `Input Sources`. With the `+` button you can add a keyboard layout for your daily usage with your mac. Please note that the keyboard layout shipped with `devonfw-ide` as example is called `German-ISO` and can be found in the `Others` section at the end of the list.



✓ Show Input menu in menu bar

☐ Automatically switch to a document's input source

When you have multiple mappings in place, on the top menu bar you will find a little icon next to the current time that allows you to switch between the keyboard layouts, what is very handy when you switch from your native MacBook keyboard to an external USB keyboard or vice versa. Even for a pure MacOS geek this can be helpful in case a friend coming from Windows/Linux is supposed to type something on the Mac in a pair-programming session.

In our German keyboard mapping example you can use the keys like `Alt Gr`, etc. to type special characters as you would expect and as printed on your keyboard. To make `Pos1`, `End`, etc. work properly accross all apps please read on to the next section(s).

In case you want to create your own keyboard layout you can of course edit the `*.keylayout` files in a text editor. However, to make this much more comfortalbe, you can use the graphical editor tool Ukelele. Besides the app itself the Ukelele `dmg` file also contains a `Documentation` and a `Resources` folder. The latter contains many keyboard layouts that you can use as a starting point.

KeyBindings

Still various keyboard shortcuts might not work as expected for you. Therefore we provide you with an advanced configuration in the folder `system/mac/keyboard/KeyBindings` that you can copy to your `~/Library` folder:

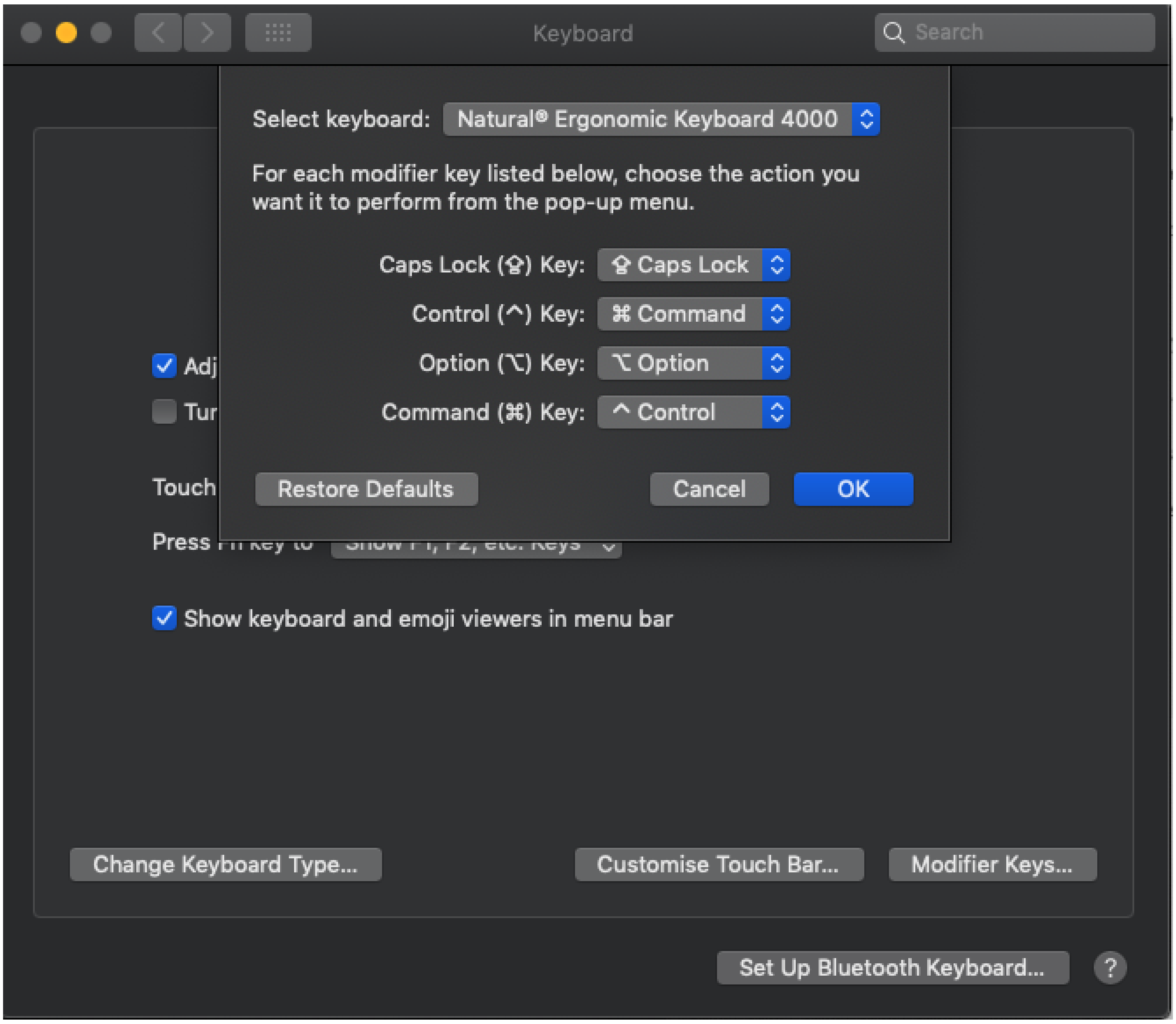
```
cd system/mac/keyboard/
cp -r KeyBindings ~/Library
```

To make the changes work you need to log-out and log-in again or you can reboot. After that your `Home` (`Pos1`) and `End` buttons should work as expected including with selection via `Shift` and/or `Command`. Also you can use `Command` together with left or right arrow key to move between words and combined with `Shift` for selection. As an example for further customization you can press `Command` + `<` to type the unicode character `«`.

However, still some apps listen to keyboard events on a lower level and come with their own keyboard mappings. In these apps you might still experience unexpected behaviour. Solutions can be found in the following sub-sections.

Switch Control and Command

If you are used to windows or linux and get easily confused by the apple keyboard behaviour you might want to switch the `Control` and the `Option` key. Open `System Preferences` and select `Keyboard`. Then in the first tab you click on the button `Modifier Keys…`. For every keyboard you can customize the behaviour of your modifier keys and therefore switch `Control` and `Option` as illustrated in the screenshot:



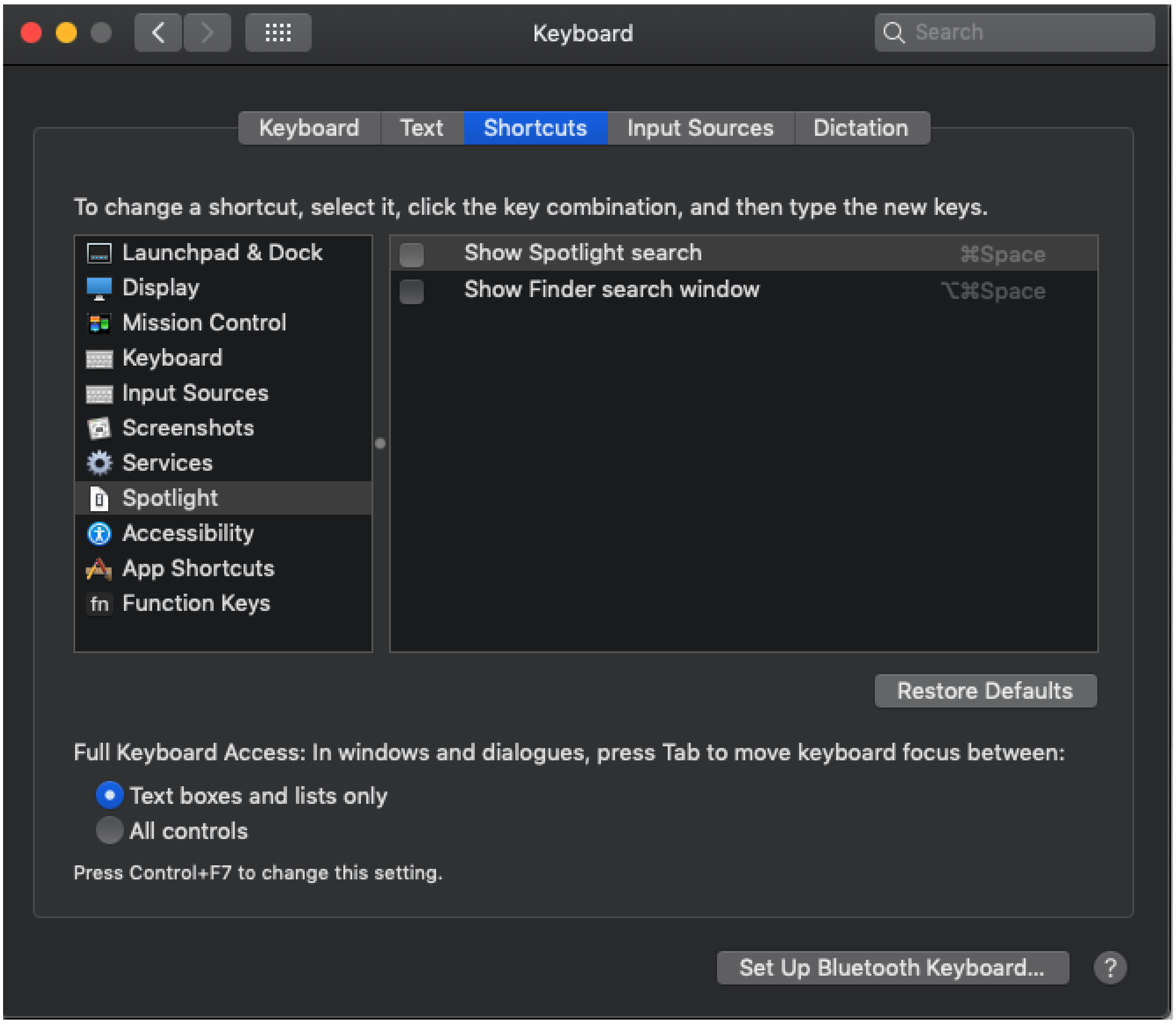
Select keyboard: Natural® Ergonomic Keyboard 4000

For each modifier key listed below, choose the action you want it to perform from the pop-up menu.

Caps Lock (⇪) Key: ⇪ Caps Lock
Control (^) Key: ⌘ Command
Option (⌥) Key: ⌥ Option
Command (⌘) Key: ^ Control

✓ Show keyboard and emoji viewers in menu bar

Programmers now should also disable that `Control` + `Space` is opening `Spotlight Search` as otherwise this shortcut can not be redifined in other apps like common IDEs.



To change a shortcut, select it, click the key combination, and then type the new keys.

☐	Show Spotlight search	⌘Space
☐	Show Finder search window	⌥⌘Space

Full Keyboard Access: In windows and dialogues, press Tab to move keyboard focus between:

[✓ Text boxes and lists only] All controls

Press Control+F7 to change this setting.

Eclipse

In Eclipse move and select by word as described above is not working. Even worse the most important shortcut does not work: `Control + Space` for code completion (content assist). You can manually redefine the keybindings in `Preferences` under `General > Keys`. However, with multiple IDE installations and workspaces this will quickly get tedious. Therefore, you can `Export` and `Import` specific `Preferences` such as `Keys Preferences` to/from a `*.epf` (Eclipse PreFerences) file. We have done all this for you so you can just import the file located in `system/mac/keyboard/Eclipse/eclipse-mac-keybindings.epf` into your Eclipse. Happy coding.

Karabiner

If you want more dynamics and do not worry about an app that has to run in the background to make your keyboard work as you like (no relevant performance overhead), you can try Karabiner Elements. This is a powerful tool to remap your keyboard shortcuts. In the UI you can only directly create and edit `Simple Modifications` that are too limited for most use-cases. However using `Complex Modifications` you can do a lot of magic to customize the keyboard behaviour to your personal needs. A key with any combination of modifiers can be mapped to any key with arbitrary modifiers. This can also be bound to conditions based on the frontmost application or the keyboard model. These complex modifications are configured as `*.json` files. We have included a set with useful rules for external keyboards, programmer shortcuts, etc. If you have Karabiner installed, you only need to copy the contents of the `karabiner` folder located in this directory to your `~/.config` folder:

```
cd system/mac/keyboard/
cp karabiner/assets/complex_modifications/*.json
~/.config/karabiner/assets/complex_modifications/
```

Now, if you open the `Complex Modifications` in the `Karabiner` app, you can click on the `+ Add rule` button and will see these mappings in the popup. Select the rules you want to add (e.g. add all) and you are done. Unlike other solutions here you can quickly tweak your keyboard without the need to logout and restart apps what gives faster trial and error turn-arounds. Furhter, if you want to tweak your own configs, Karabiner comes with a secondary app called `Karabiner-EventViewer` that shows you the names of the keys, modifiers, and apps for the events you are triggering. This is very helpful to get the config right.

8.4. Linux Tooling

Nothing here so far. If you are a Linux user, please share your experience and provide your valuable hints.

9. Support

9.1. Migration from oasp4j-ide

The `devonfw-ide` is a completely new and innovative solution for managing the local development environment that has been created from scratch. Releases of OASP as well as releases of devonfw until version 3.1.x are based on the old `oasp4j-ide` that is now considered deprecated. As `devonfw-ide` is a complete redesign this will have some impact for the users. This section should help and assist so you do not get lost.

9.1.1. Get familiar with devonfw-ide

First of all you should roughly get familiar with the new `devonfw-ide`. The key features and changes are:

- platform-agnostic (supports Windows, Mac, and Linux in a single distribution)
- small core (reduced the download package from ~2 gigabyte to ~2 megabyte)
- fast and easy updates (build in update support)
- minimum number of scripts (removed tons of end-user scripts making things much simpler)
- fully automated setup (run `setup` script and you are ready - even for advanced features that had to be configured manually before)
- single command for everything (entire CLI available via new `devon` command)

For all the details you should study the documentation starting from the beginning.

9.1.2. Migration of existing oasp4j-ide installation

- extract new `devonfw-ide-scripts` on top of your existing installation
- run `setup`
- done

If you get errors:

- ask your technical lead to fix the `settings` git repo for `devonfw-ide` or offer him to do it for you.
- you need to merge the `devon` folder into your settings
- you need to merge the `devon.properties` into your settings
- you should check your `variables[-customized][.bat]` and merge required customizations into the proper configuration

9.1.3. Hints for users after migration

Getting used to all the new commands might be tedious when starting after a migration.

Table 16. Comparison of commands

oasp4j-ide command	devonfw-ide command	Comment
`create-or-update-workspace`	`devon eclipse ws-update`	actually not needed anymore as workspace is updated automatically when IDE is launched. To launch your IDE simply run `devon eclipse`, `devon intellij`, or `devon vscode`. If you like to get launch scripts for your IDE e.g. Eclipse just call `devon eclipse --all create-script`.
`create-or-update-workspace «workspace»`	`cd «workspace» && devon eclipse ws-update`	
`update-all-workspaces`	`devon eclipse --all ws-update`	
`create-or-update-workspace-vs`	`devon vscode ws-update`	
`devcon workspace create «workspace»`	Simply create the `«workspace»` directory (e.g. `cd workspaces && mkdir examples`)	
`scripts/update-eclipse-workspace-settings`	`devon eclipse ws-reverse`	To add new properties (old option `--new`) use `devon eclipse ws-reverse-add`

`devcon project build`
`devcon devon4j build`
`devcon devon4ng build`
	`devon build`	

`devcon devon4j create`	`devon java create`	
`devcon devon4ng create`	`devon ng create`	

`devcon system *`
`devcon dist *`
	`setup` or `devon ide setup`	

oasp4j-ide command	devonfw-ide command	Comment
`console.bat`	-	Simply open terminal in selected folder. On Windows right-click folder in windows-explorer and select `open devonfw CMD here`.
`devcon help`	`devon help`	
`devcon doc`	Read the documentation from devonfw.com	

10. License

This software is licensed as Open-Source product for free usage including commercial use. You need to apply to our terms of use before using it.

10.1. Terms of Use

We provide the computer program of `devonfw-ide`, (hereinafter "Software") free of charge to download. The documentation of this software is licensed under the terms of the Creative Commons License (Attribution-NoDerivatives 4.0 International). The source code of this software itself is licensensed under the terms of the Apache License 2.0.

In accordance with the license terms, the user is entitled to use the Software free of charge (also commercially), provided that the copyright notice will be taken.

Because the software is provided free of charge, there is no warranty, to the extent permitted by applicable law. It is provided "as is" without warranties or conditions of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the software prove defective, you assume the cost of all necessary servicing, repair or correction.

The software also requires open-source components of third-parties (providers), for which the respective open source license terms apply. The open-source components of third parties are not from us and must be licensed directly from the respective third party. The rights to use will be granted directly by the respective right owner to the extent of each relevant open source license terms. The user himself can download the desired third party components from the servers of the respective provider and install them in his own environment. As the software (devonfw-ide) is also capable to download third party components automatically for you, you have to apply to this terms and the licenses of the third party components before using it.

This software integrates the following third party components (the column `inclusion` indicates whether the component is `directly included` in the download package of `devonfw-ide` we provide, it is downloaded during the `default setup`, or if it is `optional` such that it gets downloaded and installed only if explicitly triggered by the user on demand):

Table 17. Third party components

Component	Inclusion	License
devonfw-ide	Directly included	ASL 2.0 and this TERMS_OF_USE
JSON-P API	Directly included	EPL 2.0
JSON-P Implementation	Directly included	EPL 2.0
OpenJDK / AdoptOpenJDK (Java)	Default Setup	GPLv2
Maven	Default Setup	ASL 2.0
VS Code	Default Setup	MIT (Terms)
extension-pack-vscode	Default Setup	ASL 2.0

Component	Inclusion	License
Eclipse	Default Setup	EPL 2.0
CobiGen	Default Setup	ASL 2.0
TM Terminal	Default Setup	EPL 2.0 (see here)
AnyEdit	Default Setup	EPL 1.0
EclipseCS	Default Setup	LGPL 2.1
SpotBugs Eclipse plugin	Default Setup	LGPL 2.1
EclEmma	Default Setup	EPL 1.0
StartExplorer	Default Setup	WTFPL 2
regex tester	Default Setup	GPL 2.0 (see here)
eclipse-templatevariables	Default Setup	ASL 2.0
Node.js	Default Setup	License
NPM	Default Setup	Artistic License 2.0 (Terms)
Angular CLI (ng)	Default Setup	MIT
Gradle	Optional	ASL 2.0
Jenkins	Optional	MIT
SonarQube (CommunityEdition)	Optional	LGPL 3.0
SonarLint	Optional	LGPL 3+
cicdgen	Optional	ASL 2.0
devon4j	Optional	ASL 2.0
devon4ng	Optional	ASL 2.0
devon4node	Optional	ASL 2.0