



devonfw

devon-ide 3.0.0-beta21

The devonfw community

Version 3.0.0-beta21, 2019-08-27_07.14.13

Table of Contents

Introduction	1
1. Features	1
1.1. IDEs	2
1.2. Platforms	2
1.3. Build-Systems	2
1.4. Motivation	2
2. Setup	4
2.1. Prerequisites	4
2.2. Download	4
2.3. Install	4
2.4. Uninstall	4
3. Usage	6
3.1. Developer	6
3.2. Architect	6
4. Configuration	8
4.1. devon.properties	8
5. Variables	10
6. Devon CLI	12
6.1. Devon	12
6.2. Commandlets	12
6.3. build	13
6.4. eclipse	13
6.5. gradle	15
6.6. help	15
6.7. ide	15
6.8. vscode	17
6.9. java	17
6.10. jenkins	19
6.11. mvn	19
6.12. ng	20
6.13. node	20
6.14. npm	20
6.15. release	21
6.16. sonar	21
6.17. vscode	21
6.18. yarn	22
7. Structure	23
7.1. TERMS_OF_USE	23

7.2. conf	23
7.3. log	24
7.4. scripts	24
7.5. settings	25
7.6. software	27
7.7. system	28
7.8. updates	28
7.9. workspaces	28
8. Support	30
8.1. Migration from oasp4j-ide	30
9. License	32

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 [devon-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 [devon-ide](#) will solve all these problems. Here are the features we can offer for you with [devon-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 [devon-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 `devon-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 `devon-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 `devon-ide`:

```

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

Releases of `devon-ide` are published to maven central. You can download them from [here](#).

2.3. 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 (`devon-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](#).

2.4. Uninstall

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

```
devon ide uninstall
```

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

The `devon-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 resides just locally inside the folder of your `devon-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 `devon-ide` will download third party software to your `~/Downloads` folder to reduce redundant storage.

3. Usage

This section explains you how to use the `devon-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 `devon-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 `devon-ide` [scripts](#). However, you may also add the cloned settings, a custom `devon.properties` file, or predefine [software](#) (be aware of multi-plattform-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 `devon-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 `devon-ide`. Never directly modify this file!
4. `devon.properties` - vendor variables for custom distributions of `devon-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 `devon-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 `devon-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 `devon-ide` to the outside world by reducing the number of variables and only exporting those that are required.

Table 1. Variables of `devon-ide`

Variable	Value	Meaning
<code>DEVON_IDE_HOME</code>	e.g. <code>./projects/my-project</code>	The toplevel directory of your <code>devon-ide</code> structure .
<code>PATH</code>	<code>\$PATH:\$DEVON_IDE_HOME/software/java:...</code>	Your system path is adjusted by <code>devon</code> command .
<code>DEVON_HOME_DIR</code>	<code>~</code>	The platform independent home directory of the current user. In some edge-cases (e.g. in cygwin) this differs from <code>~</code> to ensure a central home directory for the user on a single machine in any context or environment.
<code>DEVON_IDE_TOOLS</code>	<code>java mvn eclipse vscode node npm ng</code>	List of tools that should be installed and upgraded by default for your current IDE.
<code>DEVON_OLD_PATH</code>	<code>...</code>	A "backup" of <code>PATH</code> before it was extended by <code>devon</code> to allow restoring it. Internal variable that should never be set or tweaked.
<code>WORKSPACE</code>	<code>main</code>	The workspace you are currently in. Defaults to <code>main</code> if you are not inside a workspace . Never touch this variable in any <code>variables</code> file.
<code>WORKSPACE_PATH</code>	<code>\$DEVON_IDE_HOME/workspaces/\$WORKSPACE</code>	Absolute path to current workspace . Never touch this variable in any <code>variables</code> file.
<code>JAVA_HOME</code>	<code>\$DEVON_IDE_HOME/software/java</code>	Path to JDK
<code>SETTINGS_PATH</code>	<code>\$DEVON_IDE_HOME/settings</code>	Path to your settings . To keep <code>oasp4j-ide</code> legacy behaviour set this to <code>\$DEVON_IDE_HOME/workspaces/main/development/settings</code> .

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

6. Devon CLI

The `devon-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 `devon-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
devon-ide has environment variables have been set for /projects/my-project
[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 may 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 `devon-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 devon-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
<code>--all</code>	if provided as first arg then to command will be invoked for each workspace
<code>setup</code>	setup Eclipse (install or update)
<code>add-plugin <id> [<url>]</code>	install an additional plugin
<code>run</code>	launch Eclipse (default if no argument is given)
<code>start</code>	same as <code>run</code>
<code>ws-up[<date>]</code>	update workspace
<code>ws-re[<verse>]</code>	reverse merge changes from workspace into settings
<code>ws-reverse-add</code>	reverse merge adding new properties

Argument(s)	Meaning
<code>create-script</code>	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)
- `devstyle`

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 his `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
<code>setup</code>	setup gradle (install and verify), configurable via <code>GRADLE_VERSION</code>
<code><<args>></code>	run gradle with the given arguments (<code><<args>></code>)

6.6. help

The `help` commandlet provides help for the CLI.

Table 4. Usage of `devon help`

Argument(s)	Meaning
	Print general help
<code><<command>></code>	Print help for the commandlet <code><<command>></code> .

Please note that `devon help <<command>>` will do the same as `devon <<command>> help`.

6.7. ide

The `ide` commandlet manages your `devon-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
<code>setup [<<SETTINGS_URL>>]</code>	setup devon-ide (cloing the settings from the given URL)
<code>update [<<package>>]</code>	update devon-ide
<code>update scripts [to <<version>>]</code>	update devon-ide
<code>uninstall</code>	uninstall devon-ide (if you want remote it entirely from your system)

6.7.1. setup

Run `devon ide setup` to initially setup your `devon-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 `devon-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 `devon-ide`. This will check for updates and `install` them automatically. The optional extrag argument (`<<package>>`) behaves as following:

- `scripts`: check if a new version of `devon-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 `devon-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 `devon-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 devon-ide installations and potential shared `software` folder. You may also want to clean up your `~/Downloads` directory from files downloaded by `devon-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 devon-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
<code>--all</code>	if provided as first arg then to command will be invoked for each workspace
<code>setup</code>	setup IntelliJ (install or update)
<code>run</code>	launch IntelliJ (default if no argument is given)
<code>start</code>	same as <code>run</code>
<code>ws-up[date]</code>	update workspace
<code>ws-re[verse]</code>	reverse merge changes from workspace into settings
<code>ws-reverse-add</code>	reverse merge adding new properties
<code>create-script</code>	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
<code>setup</code>	setup OpenJDK (install or update and verify), configurable via <code>JAVA_VERSION</code> (e.g. <code>8u222b10</code> or <code>11.0.4_11</code>)
<code>create <<args>></code>	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 <code>-DbType=<<db>></code> to choose the database (hana, oracle, mssql, postgresql, mariadb, mysql, h2, hsqldb). Any option starting with dash is passed as is."
<code>migrate [from <<version>>] [single]</code>	migrate a devon4j project to the latest version. If for some reasons the current devonfw version (e.g. <code>oasp4j:2.6.0</code>) 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."

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
<code>setup</code>	Setup Jenkins (install and verify)
<code>start</code>	Start your local Jenkins server
<code>stop</code>	Stop your local Jenkins server
<code>add-ci</code>	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 <code>MVN_BUILD_OPTS</code>
<code>setup</code>	setup Maven (install and verify), configurable via <code>MAVEN_VERSION</code>
<code>get-version</code>	Print the version of your current project. Will consolidate the version for multi-module projects ignoring <code>dev[-SNAPSHOT]</code> versions and fail on mixed versions.

Argument(s)	Meaning
<code>set-version <nv> [<cv>]</code>	Set the version of your current project to <code><nv></code> (assuming your current version is <code><cv></code>).
<code>check-top-level-project</code>	Check if you are running on a top-level project or fail if in a module or no maven project at all.
<code>release</code>	Start a clean deploy release build, configurable via <code>MVN_RELEASE_OPTS</code>
<code><args></code>	run maven with the given arguments (<code><args></code>)

6.12. ng

The `ng` commandlet ...

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 10. Usage of `devon node`

Argument(s)	Meaning
<code>setup</code>	setup node.js (install and verify), configurable via <code>NODE_VERSION</code>

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 11. Usage of `devon npm`

Argument(s)	Meaning
	run default build, configurable via <code>NPM_BUILD_OPTS</code>
<code>setup</code>	setup NPM (install and verify), configurable via <code>NPM_VERSION</code>
<code>get-version</code>	Print the version of your current project.
<code>set-version <nv> [<cv>]</code>	Set the version of your current project to <code><nv></code> (assuming your current version is <code><cv></code>).
<code>check-top-level-project</code>	Check if you are running on a top-level project or fail if in a module or no NPM project at all.
<code>release</code>	Start a clean deploy release build, configurable via <code>NPM_RELEASE_OPTS</code>

Argument(s)	Meaning
<code><<args>></code>	run NPM with the given arguments (<code><<args>></code>)

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 12. Usage of `devon sonar`

Argument(s)	Meaning
<code>setup</code>	Setup SonarQube (install and verify)
<code>start</code>	Start your local SonarQube server
<code>stop</code>	Stop your local SonarQube server
<code>add-ci</code>	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 devon-ide installation simply run: `devon vscode`

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

Table 13. Usage of `devon vscode`

Argument(s)	Meaning
<code>--all</code>	if provided as first arg then to command will be invoked for each workspace

Argument(s)	Meaning
setup	setup VSCode (install or update)
run	launch VSCode (default if no argument is given)
start	same as <code>run</code>
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 14. Usage of `devon yarn`

Argument(s)	Meaning
	run default build, configurable via <code>YARN_BUILD_OPTS</code>
setup	setup yarn (install and verify), configurable via <code>YARN_VERSION</code>
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 <code>YARN_RELEASE_OPTS</code>
<<args>>	run yarn with the given arguments (<<args>>)

7. Structure

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

File structure of your devon-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 `devon-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 [devon-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 [devon-ide](#) with a random password. This should make it easier for [devon-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 [devon-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 [devon-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 [devon-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 `devon-ide`.

7.5. settings

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

To get an initial set of these settings we provide the `devon-ide-settings` as an initial package. These are also released so you can download a the lastes stable version from `maven central`.

To test `devon-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 as technical lead creates a `settings` git repository for the project. He creates a "fork" `devon-ide-settings` by adding it as "upstream" origin and pulls from there finally pusing it to the project `settings` git. This also allows to later merge changes from the official `devon-ide-settings` back into the project specific `settings` "fork".

7.5.1. Structure

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

```
/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 `devon-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-devon-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 `devon-ide` together with the required software. However, to be platform independent and allow lightweight updates the `devon-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 publically 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 `devon-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 `devon-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 `devon-ide` installations and to prevent side-effects, it can cause a huge waste of disc resources in case you are having many installations of `devon-ide`. If you are a power-user of `devon-ide` with more than 10 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 `devon-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 `devon-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 `devon-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 `devon-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 `devon-ide` installation but let you use it for all other ones.

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 `devon-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
│   ├── / CobiGen_Templates
│   ├── / devon-ide
│   ├── / devon4j
│   ├── / my-thai-star
│   └── / sonar-devon-plugin
└── / stable
    ├── / .metadata
    ├── / devon-ide
    ├── / devon4j
    └── / sonar-devon-plugin
```

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 others just switch between those via **git checkout**.

8. Support

8.1. Migration from oasp4j-ide

The `devon-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 `devon-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.

8.1.1. Get familiar with devon-ide

First of all you should roughly get familiar with the new `devon-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 megebyte)
- 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](#).

8.1.2. Migration of existing oasp4j-ide installation

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

If you get errors:

- ask your technical lead to fix the `links:settings.asciidoc[settings]` git repo for `devon-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](#)

8.1.3. Hints for users after migration

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

Table 15. Comparison of commands

oasp4j-ide command	devon-ide command	Comment
<code>create-or-update-workspace</code>	<code>devon eclipse ws-update</code>	actually not needed anymore as workspace is updated automatically when IDE is launched. To launch your IDE simply run <code>devon eclipse</code> , <code>devon intellij</code> , or <code>devon vscode</code> . If you like to get launch scripts for your IDE e.g. Eclipse just call <code>devon eclipse --all create-script</code> .
<code>create-or-update-workspace <<workspace>></code>	<code>cd <<workspace>> && devon eclipse ws-update</code>	
<code>update-all-workspaces</code>	<code>devon eclipse --all ws-update</code>	
<code>create-or-update-workspace-vs</code>	<code>devon vscode ws-update</code>	
<code>devcon workspace create <<workspace>></code>	Simply create the <code><<workspace>></code> directory (e.g. <code>cd workspaces && mkdir examples</code>)	
<code>scripts/update-eclipse-workspace-settings</code>	<code>devon eclipse ws-reverse</code>	To add new properties (old option <code>--new</code>) use <code>devon eclipse ws-reverse-add</code>
<code>devcon project build</code> <code>devcon devon4j build</code> <code>devcon devon4ng build</code>	<code>devon build</code>	
<code>devcon devon4j create</code>	<code>devon java create</code>	
<code>devcon devon4ng create</code>	<code>devon ng create</code>	
<code>devcon system *</code> <code>devcon dist *</code>	<code>setup</code> or <code>devon ide setup</code>	
<code>devcon help</code>	<code>devon help</code>	
<code>devcon doc</code>	Read the documentation from devonfw.com	

9. License

This documentation is licensed under [Creative Commons License \(Attribution-NoDerivatives 4.0 International\)](#). The `devon-ide` software itself is licensed under [Apache Software License 2.0](#). However, it involves several third party components under different open-source licenses. You therefore have to apply our [terms of use](#) before using it.