# Task Guide

Sven van der Meer

# Table of Contents

for SKB-Framework v0.0.1, November 15, 2018

# 1. Use Tasks

## 1.1. cloc-installation

This tasks counts the lines of code of an installation. It is available in all application modes.

The task forces files that `cloc` identifies as *sh* files to be treated as *bash* files, using the `cloc` option `--force-lang="Bourne Again Shell",sh`. This ensures that the installation's include and application files (without the `.sh` extension) are counted as *bash* files.

More details on `cloc` can be found at the Github source repository on Github.

### 1.1.1. Options

The task does not have any special options.

```
-h | --help        print help screen and exit
```

### 1.1.2. Examples

Simply running the task will count the lines of code of an installation. The following is the output of running this task on the SKB-Framework in an earlier version.

```
     545 text files.
     539 unique files.
     222 files ignored.

github.com/AlDanial/cloc v 1.80  T=1.00 s (376.0 files/s, 23126.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Bourne Again Shell             226           2618           5211          12077
HTML                             1              0              1           1635
AsciiDoc                       147            308              0           1202
XML                              1              7             21             28
Ant                              1              4              1             13
-------------------------------------------------------------------------------
SUM:                           376           2937           5234          14955
-------------------------------------------------------------------------------
```

The count shows the two ANT files (build file and macro file) as ANT and XML language files. All ADOC files are shown as AsciiDoc language files The HTML file is the framework's HTML manual. The top line shows the lines of code for all *bash* scripts in the installation, including all tasks and `.id` files.

### 1.1.3. Requirements

The task requires the tool `cloc` being installed. The dependency is called by the same name: *cloc*. If `cloc` is not installed, the task will print an error.

## 1.2. repeat-scenario

This tasks repeats a scenario. It is available in all application modes.

### 1.2.1. Options

The option *scenario* should be used to identify the scenario to repeat. The scenario must be loaded. For this option, the long or short name of the scenario can be used.

The other options determine how often the scenario should be repeated (*times*) and how long the task should wait between repetisions (*wait*). Only positive integers are allowed for both options. The default value for both options is 1. So when only provided with a scenario, the task will run it once.

```
-h | --help                print help screen and exit
-s | --scenario SCENARIO   the scenario to repeat
-t | --times INT           repeat INT times
-w | --wait SEC            wait SEC seconds between repeats
```

### 1.2.2. Examples

The example below will repeat the scenario *S1* four times, waiting for 2 seconds between repetitions.

```
repeat-scenario --scenario S1 --times 4 --wait 2
```

## 1.3. wait

This tasks waits for a given amount of seconds. It can be used to *slow down* task execution, for instance in a scenario. It is available in all application modes.

### 1.3.1. Options

The option *seconds* takes a positive integer as argument for the number of seconds to wait. The default, without this option, is 1.

The actual wait time depends on the underlying system. Since *wait* is a task, a new *bash* instance is created to execute it. This creation does take time, less on powerful UNIX hosts than for instance on Cygwin or a Raspberry PI. On native UNIX systems the creation time should not be significant. The actual wait time will be printed when the task finishes.

Since *bash* does not support floating point or double integer values, only positive integers can be used.

```
-h | --help          print help screen and exit
-s | --seconds SEC   wait SEC seconds, default is 1
```

### 1.3.2. Examples

The example below will wait for 3 seconds, plus the time it takes to execute the *wait* task itself.

```
wait --seconds 3
```

# 2. Build Tasks

## 2.1. clean

This tasks cleans, i.e. removes, all artifacts that have been build by other taks. In addition to that, the directory set by the parameter *TARGET* will be removed.

The task does not actually remove any built artifacts itself. It uses the `-c` (or `--clean`) option of all tasks that do build artifacts. By convention, these are tasks whose name starts with *build-* or *compile-*.

So *clean* will lookup the loaded task map, find all tasks that start with *build-* or *compile-* and execute them with the `--clean` option. If any of these tasks fails (for instance due to missing parameters), *clean* will also fail.

### 2.1.1. Options

The option *simulate* can be used to simulate a clean. Here, not task will be executed and the directory *TARGET* will not be removed either. Instead, the task will simply print what commands it would run.

The default command to remove the *TARGET* directory is `rm -frI`. This is an additional safety feature to prevent accidental removal of the directory. The option *force* can be used to remove the directory forced, i.e. using `rm -fr` instead. *simulate* will overwrite *force*.

```
-f | --force       force mode, not questions asked
-h | --help        print help screen and exit
-s | --simulate    print only, removes nothing, overwrites force
```

## 2.2. build-manual

This tasks builds the application manual. It uses the metadata provided by the element definitions

(e.g. for a task in the task's `.adoc` file) and the general text from the application manual folder in `etc/manual`. The actual manual text comes from the AsciiDoctor files with the extension `adoc`. Element lists (e.g. for tasks or parameters) also use the description from the elements definition (in the `id` file).

Building the manual involves different steps, depending on the selected target:

- ADOC: generate an aggregated ADOC file with all text and lists.
- TEXT: convert the ADOC text into well-formatted paragraphs of plain or ANSI text, then build aggregated documents for text or ANSI formatted text. The converted text is saved in the `txt` file along with the original `adoc` file, since it is also used for the online help.
- MANP, PDF, HTML: use the aggregated ADOC file and the AsciiDoctor tool chain to generate a manpage, a single HTML file, or a PDF file. These targets require AsciiDoctor (*manp*, *html*) and AsciiDoctor-PDF (*pdf*) installed.

For all targets, the task will validate the installation to ensure that all required manual source files (*adoc* and *txt* files) are accessible. If the validation does not pass successfully, no manual artifact will be build.

The general structure is the same for all targets, i.e. the task will create the same manual just for different output formats. The structure is similar to other manual pages:

- *Name* and *Synopsis*: name, tag line, general description on how to start the application
- *Description*: a description of the application
- *Options*: a description of command line (CLI) options, including a list of options Options are further categorized as *exit options* (application will process option and exit) and *runtime options* (directing runtime behavior).
- *Parameters*: a description and list of parameters that can be used to configure the application and its tasks.
- *Tasks*: a description and list of tasks provided by the application
- *Dependencies*: a description and list of dependencies (that tasks might require)
- *Shell Commands*: a description and list of commands the interactive shell provides
- *Exit Status*: a description and list of the different exit status codes and messages.
- *Scenarios*: a description and list of scenarios provided by the application
- *Security Concerns*: remarks on potential or actual security concerns when running the application
- *Bugs*: notes on known bugs
- *Authors*: list of authors
- *Resources*: list of resources
- *Copying*: notes on copyright and other related aspects

The task then provides negative filters to exclude parts of this general structure (with the exception of *name* and *synopsis*). This allows to generate tailored manuals for any application need.

The generated targets can also be tested, i.e. shown using external applications. All text formats (adoc, plain text, ANSI formatted text, annotated text) are shown using `less`. The manpage is shown using the command `man`. HTML and PDF files are shown with a web browser and a PDF reader, respectively. Note: the parameters `BROWSER` and `PDF_READER` must be set to test these targets.

Depending on the selected targets, the task generates the following output files:

- `$APP_HOME/doc/manual/{app-name}.adoc` - aggregated ADOC file with all text for the manual
- `$APP_HOME/doc/manual/{app-name}.text` - plain text file manual
- `$APP_HOME/doc/manual/{app-name}.text-anon` - annotated plain text manual
- `$APP_HOME/doc/manual/{app-name}.ansi` - ANSI formatted text manual
- `$APP_HOME/doc/manual/{app-name}.html` - single file HTML manual
- `$APP_HOME/doc/manual/{app-name}.pdf` - single file PDF manual
- `$APP_HOME/man/man1/{app-name}.1` - the generated manpage

In *warning* and *info* level, the task does not output any information (except errors). In *debug* level, the task provides detailed information about the progress. Build all targets, including the ADOC to text transformation, can take a few minutes even on a powerful host.

### 2.2.1. Configuration

The task can be configured with two parameters:

- *SKB_FW_TOOL* - optional, to find the tool for ADOC to text conversion, if not set, the target *src* cannot be build. This also requires the dependency *jre* to execute the tool.
- *MANUAL_SRC* - optional, to set the source directory for the application related parts of the manual. If used, it must be a directory, readable, and with the correct source files (to pass validation).

### 2.2.2. Manual Source

The general text for the manual is located in the directory pointed to by the parameter *MANUAL_SRC*. This directory must have the following layout and contents:

```
MANUAL_SRC
    ├──────── application                ①
    │       ├──────── authors.adoc       ②
    │       ├──────── bugs.adoc          ③
    │       ├──────── copying.adoc       ④
    │       ├──────── description.adoc   ⑤
    │       ├──────── resources.adoc     ⑥
    │       └──────── security.adoc      ⑦
    ├──────── elements                   ⑧
    │       ├──────── commands.adoc      ⑨
    │       ├──────── dependencies.adoc  ⑩
    │       ├──────── exit-options.adoc  ⑪
    │       ├──────── exit-status.adoc   ⑫
    │       ├──────── options.adoc       ⑬
    │       ├──────── parameters.adoc    ⑭
    │       ├──────── run-options.adoc   ⑮
    │       ├──────── scenarios.adoc     ⑯
    │       └──────── tasks.adoc         ⑰
    └──────── tags                       ⑱
            ├──────── authors.txt        ⑲
            └──────── name.txt           ⑳
```

① Directory with general application text, the `.adoc` source must be present, the `.txt` files can be generated using the target *src*. For multi-paragraph text, use an empty line to separate paragraphs. To add a list, add an empty line and then each list element in a single line starting with an asteriks *. Finish the list with an empty line. Nested lists are not supported.

② List of authors.

③ Statements on bugs, known problems, etc.

④ Statements on copyright, licenses, etc.

⑤ A description of the application.

⑥ Links to resources, for instance a source repository or issue management or a web site.

⑦ Statements on security concerns when using the application.

⑧ Directory with text for the framework elements, the `.adoc` source must be present, the `.txt` files can be generated using the target *src*. For the ADOC source, the same rules as for the *application* directory apply.

⑨ Introduction to shell commands.

⑩ Introduction and text for dependencies.

⑪ Text for *exit* command line options.

⑫ Text for exit status codes and error messages.

⑬ Introduction and text for command line options.

⑭ Introduction and text for parameters.

⑮ Text for *runtime* command line options.

⑯

Introduction and text for scenarios.

⑰ Introduction and text for tasks.

⑱ Directory with tags, these files are used as plain text files.

⑲ A list of authors, used in the ADOC file header.

⑳ A tag line for the application, used in the *name* and *synopsis* sections.

### 2.2.3. General Options

Following the SKB-Framework convention, the task has two main options: *clean* to remove built manual artifacts and *build* to build them. When *build* is used, other general options and filters can be used to direct the build:

- *all* - build everything (*src*, primary targets, secondary targets)
- *primary* - build all primary targets, i.e. *src* and *adoc*
- *secondary* - build all secondary targets, i.e. *text* (plain, ANSI, annotated), *manp*, *html*, and *pdf*

```
-A | --all         set all targets, overwrites other options
-b | --build       builds a manual (manpage), requires a target
-c | --clean       removes all target artifacts
-h | --help        print help screen and exit
-p | --primary     set all primary targets
-s | --secondary   set all secondary targets
-t | --test        test a manual (show results), requires a target
```

### 2.2.4. Target Options

Targets can also be selected individually. The target options can be used in any sequence in the command line, the task will automatically generate all manual artifacts in the correct order. For the secondary targets that require *adoc* to be build, the task will also automatically generate *adoc* if the file does not exist. Text sources (target *src*) are not created automatically, only on request.

```
--adoc        secondary target: text versions: ansi, text
--html        secondary target: HTML file
--manp        secondary target: man page file
--pdf         secondary target: PDF file)
--text        secondary target: text versions: ansi, text
--src         primary target: text source files from ADOC
```

### 2.2.5. Element List Filters

Some parts of the manual list application elements. For selected element types, the element list filters can be used to direct what these lists contain:

- *loaded* - applies to task lists and scenario lists. If not used, all tasks and all scenarios will be listed. If this option is used, only the loaded tasks and scenarios are listed. Loaded here means

that the elements are defined for mode the application was started in *and* have been successfully loaded.

- *requested* - applies to dependency lists and parameter lists. If not used, all dependencies and all parameters will be listed. If this option is used, only the requested dependencies and parameters are listed. Requested here means any dependency or parameter requested by a loade task.

```
-l | --loaded      list only loaded tasks and scenarios
-r | --requested   list only requested dependencies and parameters
```

### 2.2.6. Application Filters

These filters are negative filters to exclude general (application related) parts of the manual. The option name corresponds to the heading in the general manual structure described above. The default behavior is to include all general parts.

```
--no-authors           do not include authors
--no-bugs              do not include bugs
--no-copying           do not include copying
--no-resources         do not include resources
--no-security          do not include security
```

### 2.2.7. Element Filters

These filters are negative filters to exclude element description or element lists. By default, all element descriptions and all element lists are included in the manual. To no show a list, use the *no-*-list* options. To now show any description for an element type, use the *no-** options (without *-list*).

To give an example: to show all information about tasks do not use any of these filters. To show the general text, but no task list, use *--no-task-list*. To no show any information about tasks, use *--no-tasks*.

```
--no-commands          do not include commands
--no-command-list      include command text, but no list
--no-deps              include dependency text, but no list
--no-dep-list          do not include dependencies
--no-exitstatus        do not include exit status
--no-exitstatus-list   include exit status text, but no list
--no-options           do not include options
--no-option-list       include option test, but no list
--no-params            do not include parameters
--no-param-list        include parameter text, but no list
--no-scenarios         do not include scenarios
--no-scenario-list     include scenario text, but no list
--no-tasks             do not include tasks
--no-task-list         include task text, but no list
```

### 2.2.8. Examples

The following example will use the framework tool to convert *adoc* sources into well-formatted plain text.

```
build-manual --build --src
```

The following examples builds the targets *adoc*, *text*, *manp*, *html*, and *pdf*. All tasks and scenarios will be listed. Only requested dependencies and parameters will be listed.

```
build-manual --build --requested --adoc --text --manp --html --pdf
```

### 2.2.9. Task Requirements

The task has the following requirements:

- *SKB_FW_TOOL* - optional, to find the tool for ADOC to text conversion, if not set, the target *src* cannot be build. This also requires the dependency *jre* to execute the tool.
- *MANUAL_SRC* - optional, to set the source directory for the application related parts of the manual. If used, it must be a directory, readable, and with the correct source files (to pass validation).
- *asciidoctor* - optional dependency required to generate *manp* and *html* targets. If it does not exist, these targets cannot be generated.
- *asciidoctor-pdf* - optional dependency required to generate the *pdf* target. If it does not exist, this target cannot be generated.
- *start-browser* - optional task to start a web browser testing the generated *html* target. If not present,not successfully loaded, or has missing parameters, the target *html* cannot be tested.
- *start-pdf-reader* - optional task to start a PDF reader testing the generated *pdf* target If not present,not successfully loaded, or has missing parameters, the target *pdf* cannot be tested.

The task will automatically test of the required directories exist. If not, they need to be created manually, since the task does not create any directories:

- `$APP_HOME/man/man1` - for the *manp* target
- `$APP_HOME/doc/manual` - for all other targets

A few standard framework tasks are also required (all of them are mandatory and included in a standard framework installation):

- *describe-option* - this task is used to generate option lists.
- *describe-parameter* - this task is used to generate the parameter list.
- *describe-task* - this task is used to generate the task list it is a mandatory.
- *describe-dependency* - this task is used to generate the dependency list.

- *describe-existatus* - this task is used to generate the exit status list.
- *describe-command* - this task is used to generate the command list.
- *describe-scenario* - this task is used to generate the scenario list.
- *validate-installation* - this task is used to validate all input required for the manual, i.e. *adoc* and *txt* files

## 2.3. build-mvn-site

This task will build Maven sites. It uses Apache Maven for the build process. The sites need to be provided by he parameter `MVN_SITES`. Several options are available to control the build process.

### 2.3.1. Configuration

### 2.3.2. General Options

```
-b | --build            builds site(s), requires a target and site ID or all
-c | --clean            cleans all site(s)
-h | --help             print help screen and exit
-l | --list             list sites
-T | --test             test sites, open in browser
```

### 2.3.3. Target Options

```
-t | --targets          mvn: all targets
     --ad               mvn: site:attach-descriptor
     --site             mvn: site
     --stage            mvn: site:stage
```

### 2.3.4. Filter Options

```
-A | --all              all sites
-i | --id ID            site identifier for building
```

### 2.3.5. Maven Options

```
--profile PROFILE  mvn: use profile PROFILE
```

### 2.3.6. Requirements

param MVN_SITES opt

dep maven opt

task start-browser opt

# 3. Development Tasks

## 3.1. set-file-versions

This task changes version information in source file headers. It runs Apache ANT using a simple build script, which in turn calls a macro that changes the version line. The default build and macro files change java files and all relevant framework files. These files can be used as template for writing other substitutions, if required.

```
-b | --build-file FILE   ANT build file
-d | --directory DIR     start directory with source files
-h | --help              print help screen and exit
-m | --macro-file FILE   ANT macro file
-v | --version VERSION   new version string
```