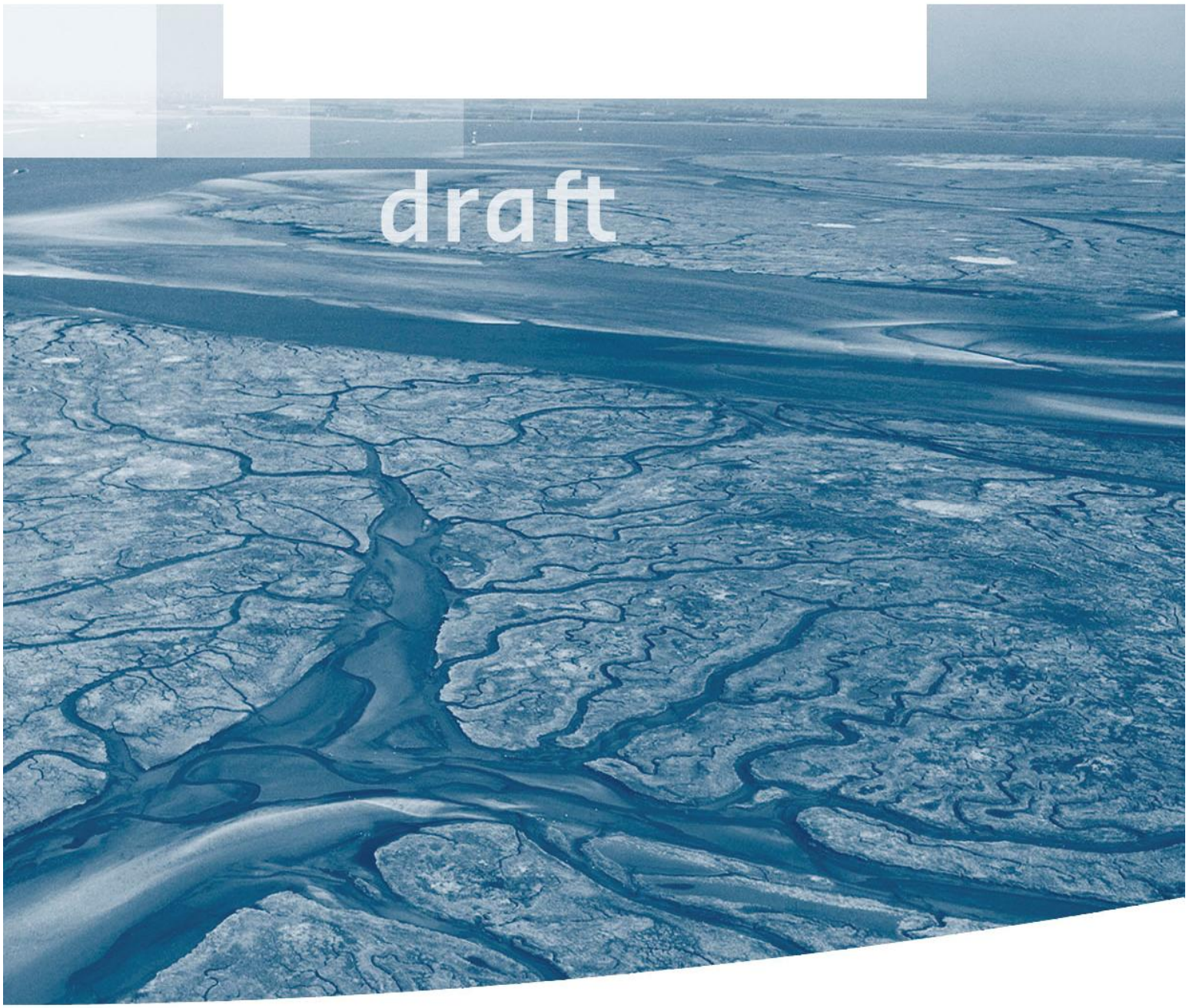
Deltares

Enabling Delta Life

Skillbed

Documentation and Manual



Deltares

Skillbed

Documentation and Manual

Bas Hoonhout

Deltares

Title
Skillbed

Pages
1

Keywords
Place keywords here

Summary
Place summary here

References
Place references here

Version	Date	Author	Initials	Review	Initials	Approval	Initials
	nov. 2011	Bas Hoonhout					

State
draft
This is a draft report, intended for discussion purposes only. No part of this report may be relied upon by either principals or third parties.

Contents

1 Introduction

The skillbed is a universal framework for monitoring the qualitative performance of numerical models. Series of characteristic model configurations are executed using a specific version of the numerical model. The model results are analyzed using generic analysis methods and the results of the analysis are distributed in a user-friendly and pro-active way. In brief, this is the test cycle facilitated by the skillbed.

The skillbed is a test environment. Nowadays, testing is common use in model development. However, these tests are often functional. They compare the model code with the model design. Functional tests are performed on different levels, from unit tests to acceptation tests.

It is important that the model code functions as designed. However, comparing model and design does not provide any insight in the quality of the design itself. This is where the skillbed comes at hand. As illustrated in Figure 1.1, real-world measurements lead to data. At some point, a model design is formulated based on some kind of research, often using measurements. The model design is translated into model code, containing modules, functions, et cetera. All modules and functions provide certain functionality. These functionalities are tested using unit tests. If necessary, the code is adapted to ensure that the code parts function as designed. Finally, the code is submitted to an overall functionality test: the acceptance test. If the code does not pass the acceptance test, the code is adapted again and the test cycle starts over. If the code passes the acceptance test, the model is ready to use.

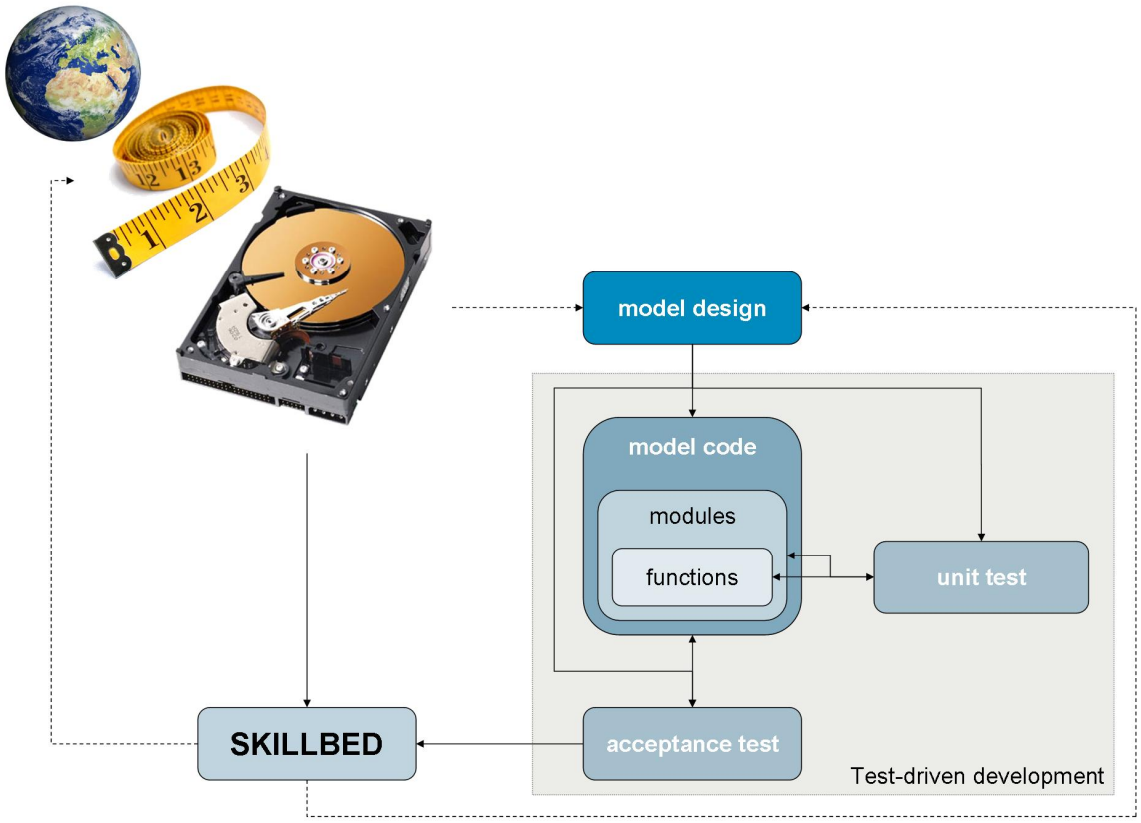


Figure 1.1 Position of Skillbed in model development cycle

Or is it? The model code now sure performs as the model design prescribes. However, at no particular point in the development of the model, a comparison is made with the original data. This is what the skillbed does. Based on the skillbed results, either the model design can be adapted and the development cycle starts over again, or new research and measurements can be initiated, or both.

A major difference between regular test results and the skillbed test results is that the skillbed does not provide definite answers on the skill of the numerical model. It only facilitates expert judgments by generating statistical data on the model-data comparison. Where a model code should perform exactly as the model design describes, a model code is hardly ever expected to reproduce measurement data exactly. Measurements are always troubled by various errors and the model design is often only a simplification of the real-world. An exact match will therefore never happen and expert judgment is still necessary to approve the model results. Again, the skillbed only facilitates this approval.

This document is both a manual and technical documentation of the skillbed. The technical documentation is mainly found in the last chapter on code structures. All other chapters may be used as manual.

2 Components

The skillbed is a client/server system. This means that the system consists of two components: a client and a server. A skillbed run is started using the client. A single client can use multiple servers and multiple clients can use the same server. The result is that the skillbed can be started on any simple desktop PC without requiring many resources. The hard work is done on the servers. At the same time, the hard work can be divided over multiple servers, thus limiting the runtime. The components of the skillbed, there are actually three, are described in more detail in the following sections and Figure 2.1.

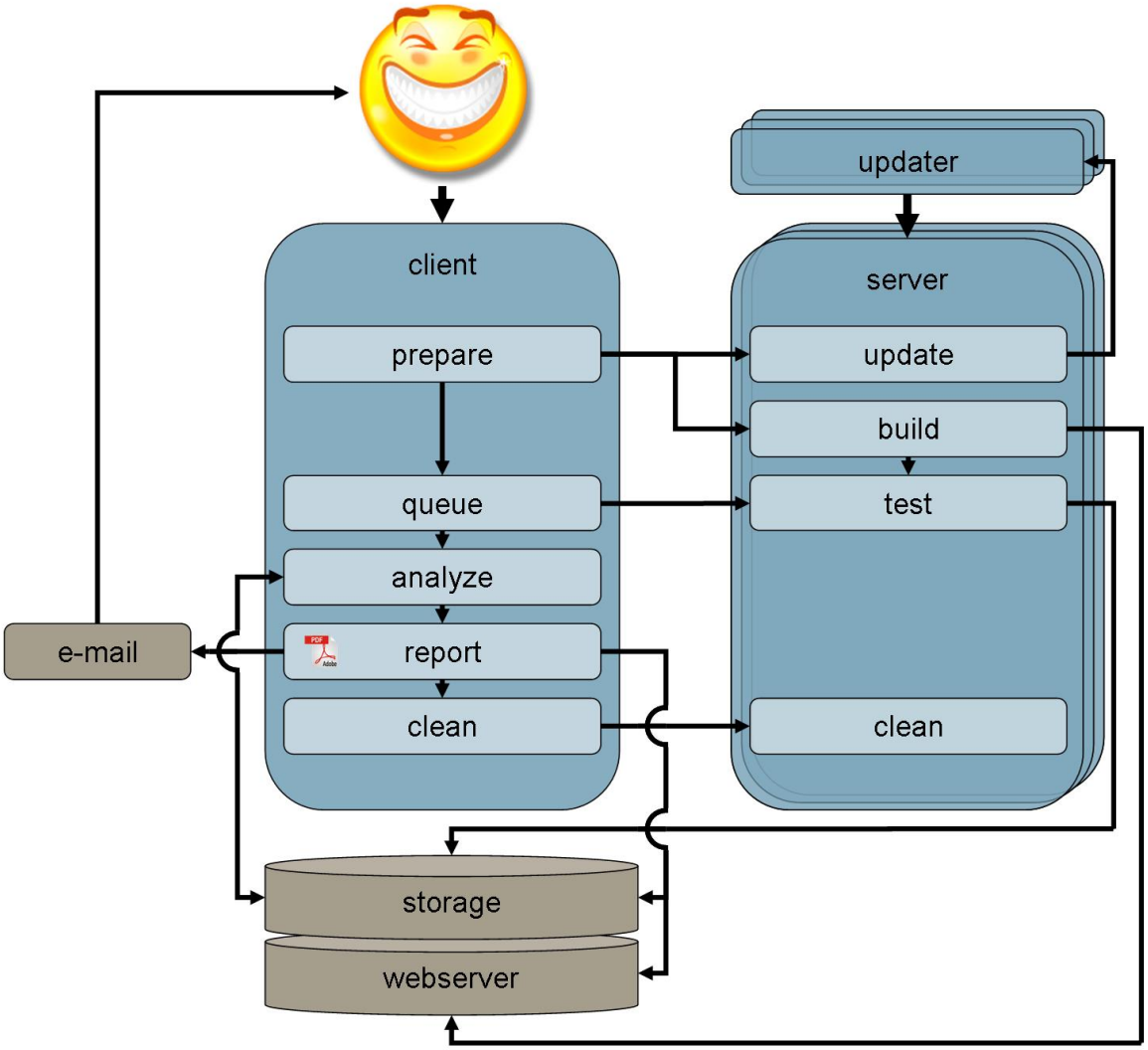


Figure 2.1 Skillbed structure

2.1 Client

The component used by the end-user is the skillbed client. Using the skillbed client, a skillbed run can be initiated. The client prepares the run by updating all servers and let them build one or more executables from the latest code. Each executable necessary is built by each of the servers. Subsequently a test queue is created that starts the model runs in an optimized order on any of the servers available. Each server uses the executables built on that specific server.

Once all tests are finished, the results from each server are copied to a central storage location from which the client copies all results to its local drive. The results are analyzed and visualized and a report is generated. The report is distributed by e-mail to the end-user and published on a webserver. The local drive of the client is cleaned as are the drives on each server.

Any data produced by the client, like analysis scripts and results, reports, log files, et cetera is stored on the central storage location.

2.2 Server

A server runs continuously waiting for a client request. When a server is started, it registers itself on a central storage location so the clients can find it. The server determines its resources and operating system and provides this information to clients upon requests. It also provides information on the number of processes running upon request.

When a client prepares a skillbed run, it notifies the servers that are about to be used to be prepared as well. Subsequently, it distributes test runs over those servers. The server runs the test and registers the process. The client continuously polls if the distributed processes are finished. Once they are, the process is unregistered and resources come available for the same or other clients.

Any data produced by the server, like executables, test results, log files, et cetera is stored on the central storage location.

2.3 Updater

The updater is like a shell around the server. In fact, instead of starting a server, in a production environment the updater should be started. The updater subsequently starts the server. The updater therefore takes the same arguments as the server.

The advantage of the updater shell is that the source code of the skillbed itself can be reloaded using a client. On request, the server tells the shell to reload the server. The updater then simply unloads all server modules and reloads them again. The updater facilitates the development of the skillbed itself. Each server runs in its own updater shell.

3 Installation

3.1 Requirements

You can run the skillbed on a single PC, which will then be both the client and the server. In order to distribute the workload over multiple PC's, you will need multiple PC's and a Subversion (SVN) server for the distribution of the configuration files and version management.

The PC's need to have some additional software installed in order to run the skillbed. This software is listed in Table 3.1.

Table 3.1 Required software

Name	Website	Component	Platform
Python	http://www.python.org/	All	All
PySVN	http://pysvn.tigris.org/	All	All
Pylons	http://pylonshq.com/	Client	All
Mako	http://www.makotemplates.org/	Client	All
Miktex	http://miktex.org/	Client	Windows
pdflatex		Client	Linux
Matlab	http://www.mathworks.nl/	Client	All
MPICH2	http://www.mcs.anl.gov/	Server	All
MS Visual Studio 2008	http://www.microsoft.com/	Server	Windows
Intel Fortran Compiler	http://software.intel.com/	Server	Windows
GNU Fortran Compiler	http://gcc.gnu.org/fortran/	Server	Linux

The skillbed requires the environment variables listed in Table 3.2 to be set. These environment variables can also be set through the *skillbed.inst* file in the *tools* directory.

Table 3.2 Required environment variables

Name	Value	Component	Platform
PYTHON_PATH	Path to python executable	All	All
PDFLATEX_PATH	Path to pdflatex executable	Client	All
BIBTEX_PATH	Path to bibtex executable	Client	All
MATLAB_PATH	Path to Matlab executable	Client	All
MPIEXEC_PATH	Path to mpiexec executable	Server	All
VS90COMNTOOLS	Microsoft Visual Studio 2008 installation directory	Server	Windows
IFORT_COMPILER11	Intel Fortran Compiler installation directory	Server	Windows
INTEL_LICENSE_FILE	Path to Intel Fortran Compiler license file	Server	Windows
GFORTRAN_PATH	Path to gfortran executable	Server	Linux

3.2 Download

The skillbed source code is available through the OpenEarthTools (OET) Subversion (SVN) repository. This is a free and open-source collection of tools for a variety of water related disciplines. See for more information http://www.openearth.eu/.

You can request a free username for the OpenEarthTools repository via the Deltares Open-Source Software (OSS) webpage at http://oss.deltares.nl/.

The skillbed source code can be found at https://svn.oss.deltares.nl/repos/openearthtools/trunk/python/applications/skillbed/. Technically, only the *tools/python/skillbed/* directory contains actual source code. The other directories contain a dummy configuration for a specific skillbed installation.

3.3 Installation

The skillbed Subversion location contains both a dummy configuration for your skillbed installation, which you can use as starting point for your own configuration, and the skillbed source code, which is generic. In order to create a new skillbed installation, you will need to make an export of the skillbed Subversion location. You need to make an export instead of a checkout, because you will modify the skillbed configuration to your needs, after which it will not be generic anymore. You should then commit your skillbed configuration to another Subversion server and/or location of your choice.

In contrast to the skillbed configuration, the skillbed source code is generic. In order to obtain updates of the source code automatically, the source code should be linked to the original skillbed Subversion location. This can be done using an *external*.

The file *tools/skillbed.inst* is machine dependent and should not be committed to the repository as well. Just leave it as an ignored file in the *tools* directory.

All steps for the installation are described in the following subsections. The Subversion client TortoiseSVN is used in this example (http://www.tortoisesvn.net/).

3.3.1 Create a checkout

Start with creating a checkout of the location where your skillbed installation should be stored (Figure 3.1). You only need to checkout a single directory, if you want. It is also possible to skip this step and use the *Import* function to store your skillbed configuration in the right location afterwards. In that case, you still need to create a checkout afterwards if you want to have a working copy on your current machine.

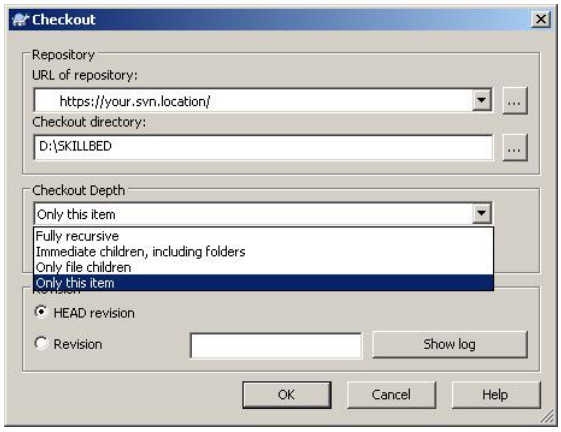


Figure 3.1 Create a checkout

3.3.2 Create an export

Now create an export of the skillbed dummy configuration (and source code) in your newly created working copy by opening the *Repo-browser* (Figure 3.2). Type in the location of the skillbed Subversion location and and select *Export…* (Figure 3.3). Select the location of your newly created working copy and click *OK* (Figure 3.4). The skillbed software will be downloaded (Figure 3.5).

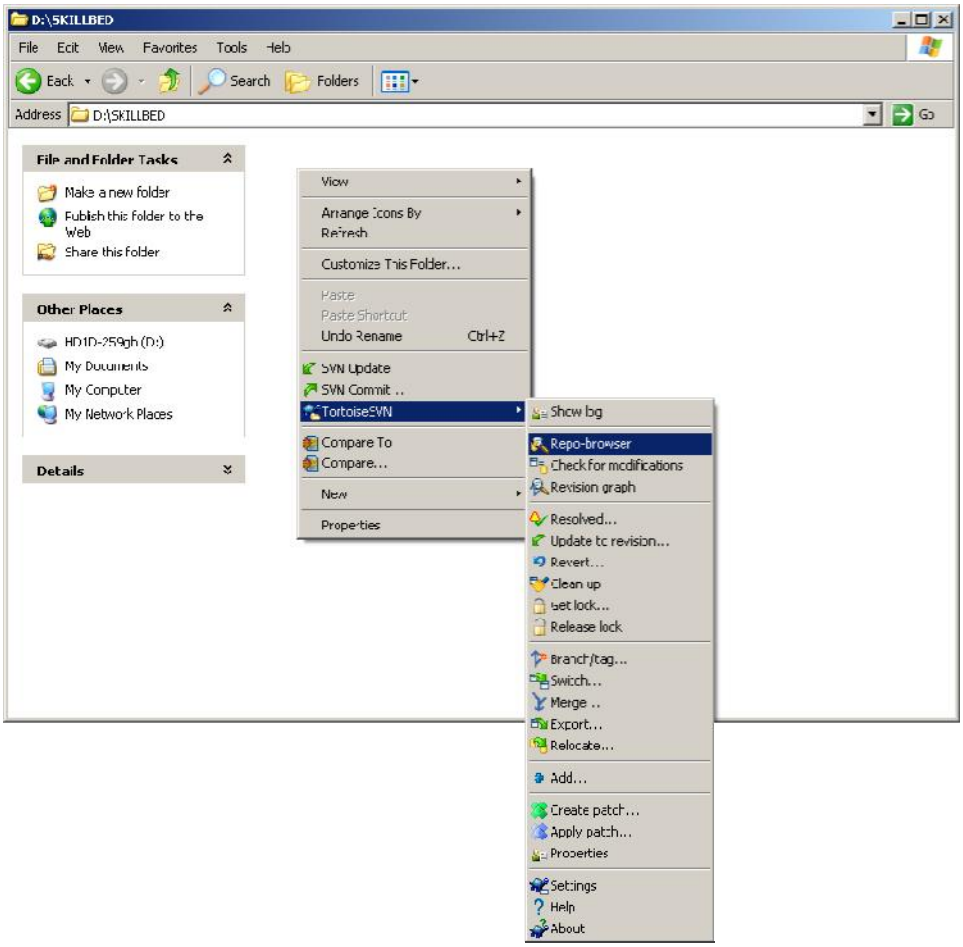


Figure 3.2 Open the Repo-browser

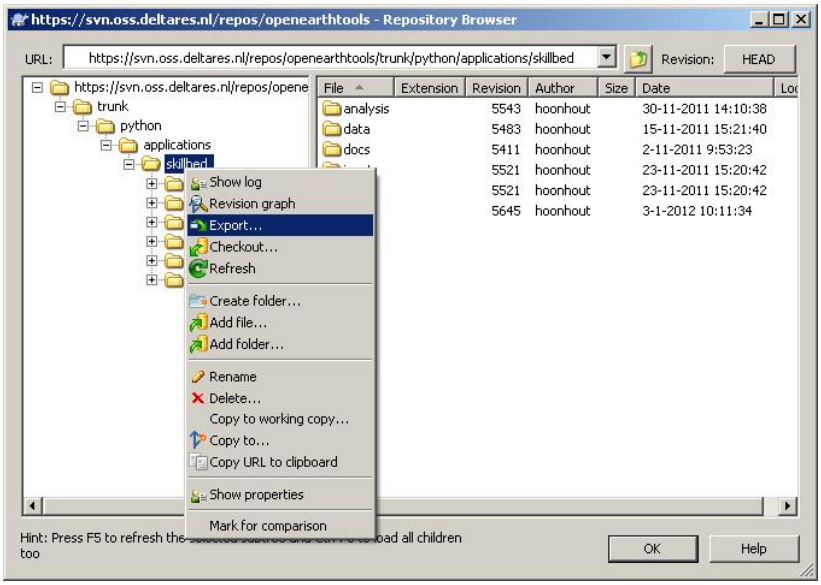


Figure 3.3 Create an export

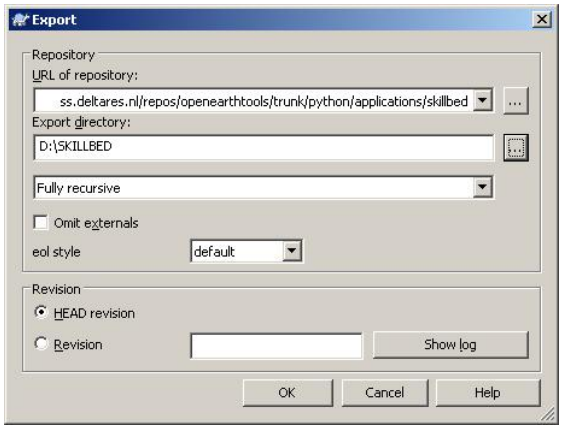


Figure 3.4 Select export location

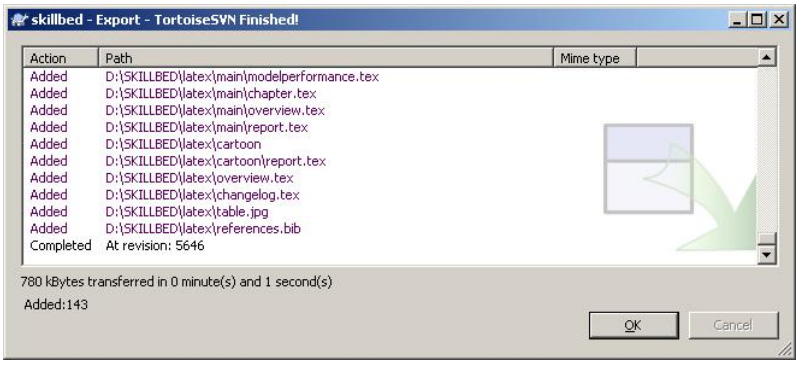


Figure 3.5 Export succeeded

3.3.3 Create an external

Now you obtained a full copy of the dummy configuration and source code of the skillbed. For the configuration part, this is a good thing. For the source code part, it isn't. Therefore, remove the directory containing the skillbed source code (Figure 3.6) and add all directories left, execpt for the *tools/skillbed.inst* file, to your new Subversion location (Figure 3.7). Next, create a new property for the source code (*python*) directory (Figure 3.8). Select *New…* (Figure 3.9) and subsequently select *svn:externals* (Figure 3.10). Use the entire address to the original source code directory plus the directory name *skillbed*, separated by a space, as value for the property (Figure 3.12). Click *OK* and the property will be added (Figure 3.11). Again, click *OK* and the link to the original Subversion location for the source code will be restored.

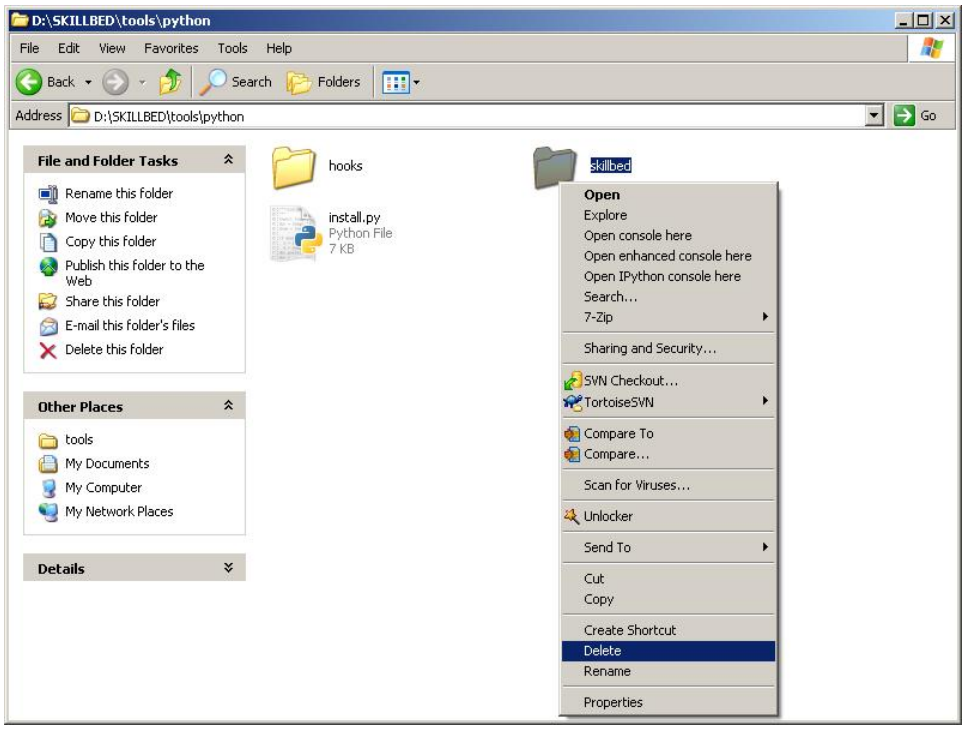


Figure 3.6 Delete exported directory containing source code

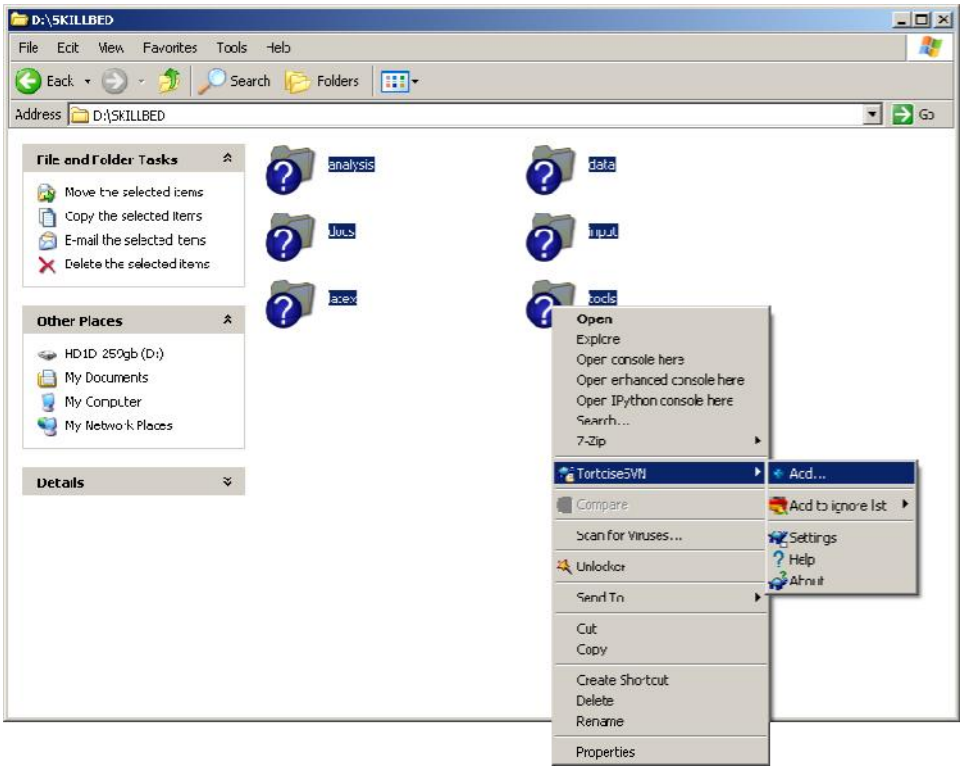


Figure 3.7 Add other directories to Subversion repository location for the skillbed

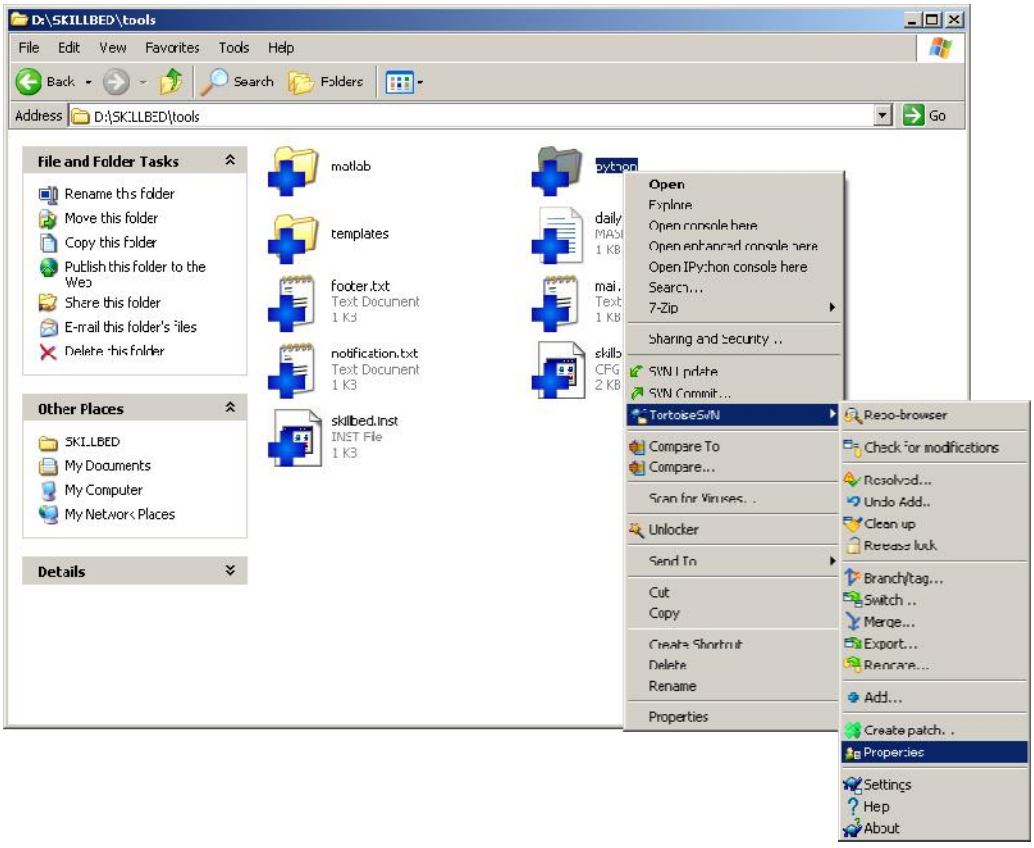


Figure 3.8 Create property for the source code (python) directory

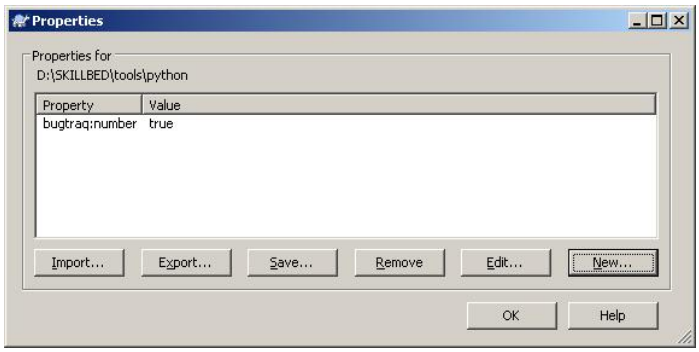


Figure 3.9 Create new property item

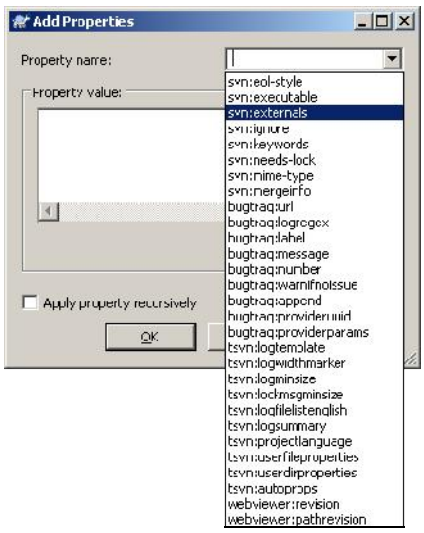


Figure 3.10 Select external type

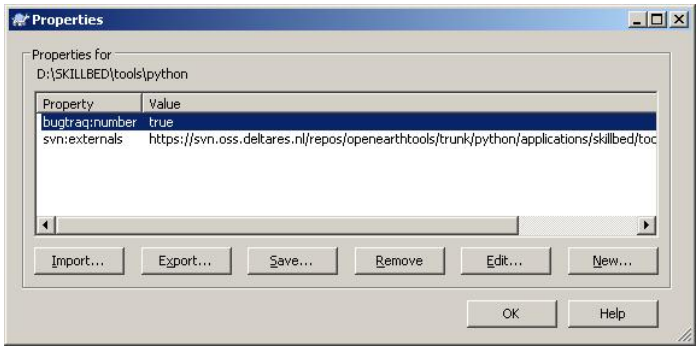


Figure 3.11 External defined

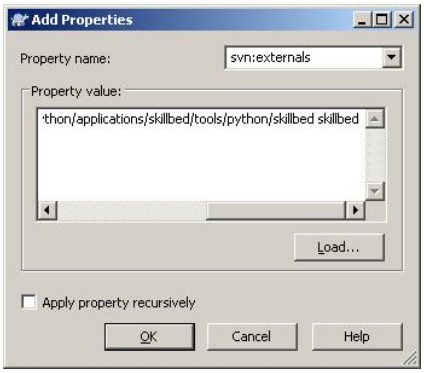


Figure 3.12 Define external

3.3.4 Commit your installation

Your initial skillbed configuration is finished. Commit your changes to your skillbed Subversion location (Figure 3.13, Figure 3.14). In order to restore the skillbed source code in your working copy, update your working copy (Figure 3.15).

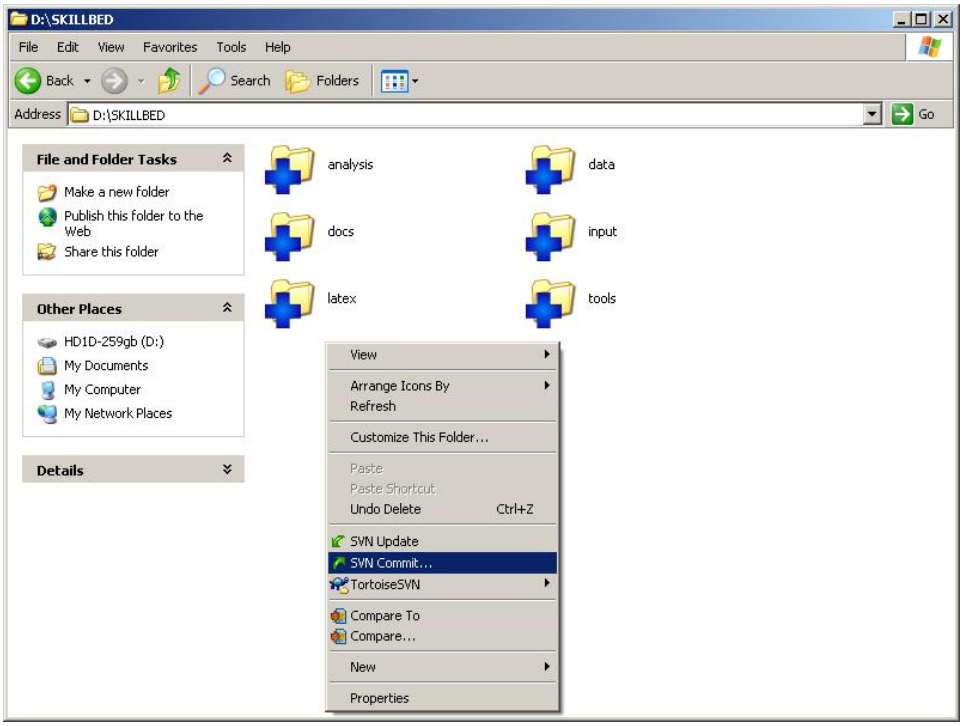


Figure 3.13 Commit initial skillbed configuration

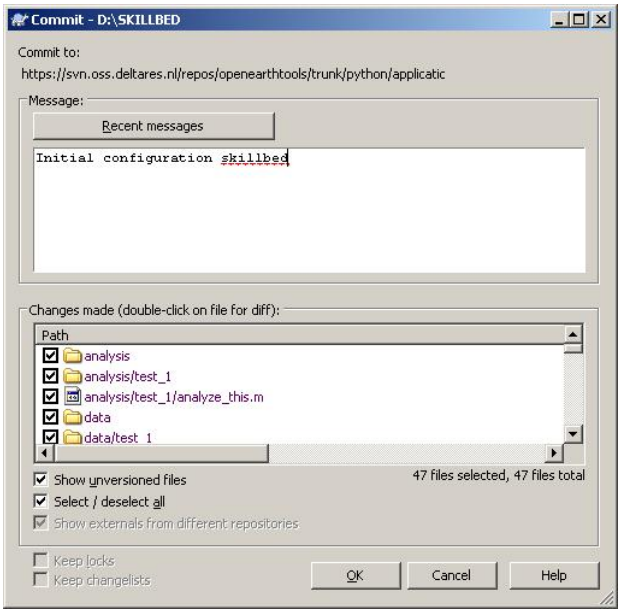


Figure 3.14 Add comments to commit

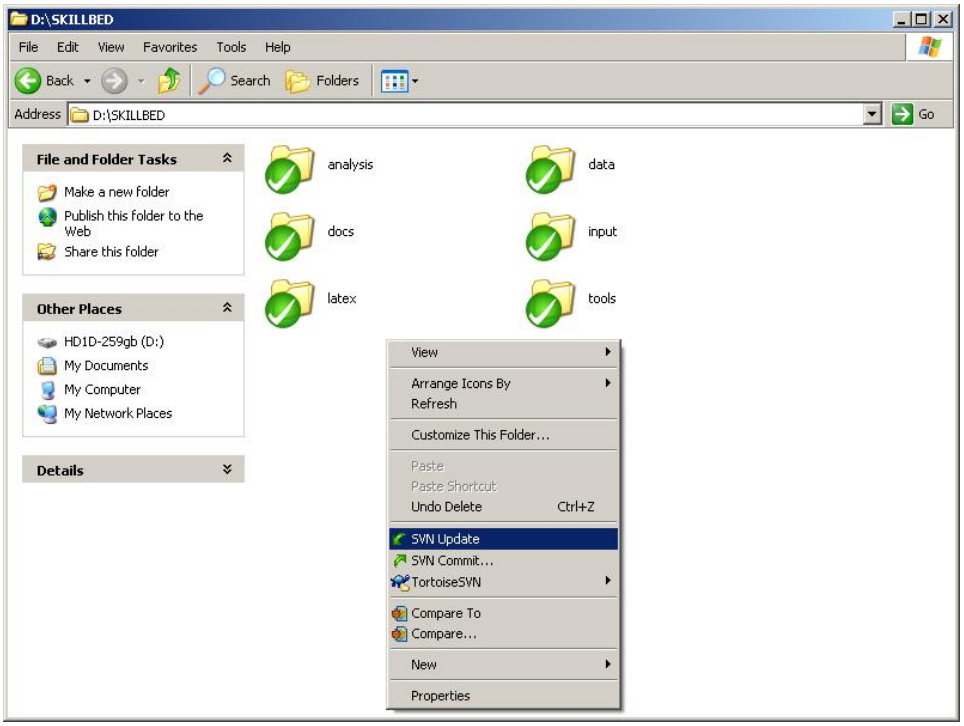


Figure 3.15 Update working copy to restore source code from original Subversion location

3.4 Files and directories

The dummy configuration contains the initial directory structure of the skillbed, which is explained in this section. The directories can be subdivided in four categories:

1. Skillbed source code and global configuration (tools)
2. Test configuration (input and data)
3. Analysis configuration (analysis and latex)
4. Documentation (docs)

3.4.1 tools

The tools directory contains the skillbed itself. This is the source code and the global configuration of the skillbed. The global configuration defines how the skillbed should be run, for example on which servers.

The tools directory contains a few configuration files that are explained in 4.1 Global configuration. It also contains three directories:

1. matlab
2. python
3. templates

The matlab directory will be empty initially, but can contain additional matlab functions used in the model result analysis. The python directory contains the skillbed source code, including possible hooks (see 7.4 Hooks). The templates directory contains two types of templates: Matlab and HTML templates (see 7.3 Templates).

3.4.2 input and data

The input directory contains the test configurations. The directory can contain an arbitrary directory substructure. In this substructure, the skillbed looks for *.config* files. Any directory containing such file is expected to contain one or more tests defined in this configuration file (see 4.2 Test configuration) and is called a *test directory*. The name of the test is defined as the path from the *input* directory to the test directory, where slashes (\ or /) are replaced by underscores (_). For example: the file *input/collection1/test2/.config* corresponds to test *collection1_test2*.

A single test configuration can contain multiple model configurations, which are called *runs*. Each run should be configured in the *.config* file and be stored in its own subdirectory. Files that should be available to all runs can be placed in the test directory itself.

The *data* directory is originally designed as the storage location for measurement data. This data can also be placed in the *input* directory along with the model configuration, but separation of measurements and models might be useful. The *data* directory is available from the analysis scripts and should only contain directories that correspond to full test names, so where the slashes are replaced by underscores.

3.4.3 analysis and latex

The *analysis* directory is similar to the data directory. It contains the Matlab scripts for the model result analysis (see 4.2.1 Analysis configuration). The scripts can also be placed in the *input* directory, if preferred. In contrast to the *data* directory, the *analysis* directory can be subdivided in multiple folder levels like the *input* directory, although this is not obligatory. Any partial match with the test name or test plus run name will work.

The *latex* directory contains the blueprints of the final reports that are produced and distributed by the skillbed. Multiple reports can be defined in multiple subdirectories. The *_tests* directory, however, is not a report configuration, but can contain text snippets that can be used throughout the reports. This directory can be used to store independent texts describing tests.

3.4.4 doc

The documentation directory contains this manual and possible other documentation files.

4 Configuration

The configuration of the skillbed is done at two levels: globally and for individual tests. The global configuration determines how and where the skillbed is run, but also provides some default settings for the individual tests. These default settings and more can be configured using the test configurations.

Configuration options are different from command line options, since command line options are restricted to a single skillbed run. Command line options may overwrite configuration options, though. This chapter explains all configuration options. The command line options are explained in the next section.

4.1 Global configuration

The main global configuration is divided over two files: *skillbed.inst* and *skillbed.cfg*. The *skilbed.inst* file is basically a list of environment variables. The variables can either be set as machine environment variables or be set in this file, which is loaded in the machine environment during the initialization of the skillbed. The variables are machine dependent. This file should therefore never be committed to the Subversion repository. The environment variables that can be set using this file are listed in Table 4.1.

Table 4.1 Environment variables

Name	Value
PYTHON_PATH	Path to python executable
PDFLATEX_PATH	Path to pdflatex executable
BIBTEX_PATH	Path to bibtex executable
MATLAB_PATH	Path to Matlab executable
MPIEXEC_PATH	Path to mpiexec executable
SVN_USERNAME	Username to be used to access the SVN repository (read-only)
SVN_PASSWORD	Password accompanying the SVN username
SCP_USERNAME	Username to be used to access the SCP server
SCP_PASSWORD	Password accompanying the SCP username

The files *skillbed.cfg* contains the machine independent part of the global configuration. It overwrites the values in the *tools/python/skillbed/config/skillbed.def* file. The file has an INI structure that consists of several sections indicated with a header between brackets ([and]). Each section has a collection of name/value pairs that are explained in Table 4.2.

Some sections do not consist of a pre-defined number of options. For example, the number of reports that can be generated is infinite. Each option in the *reports* section defines a new report with the name equal to the name of the name/value pair used. The same holds for network paths, repositories, binaries and recipients.

You can create a global configuration with a name different from *skillbed.cfg*. In order to use such file, you will need to specify its name as command line argument, as will be explained in the next chapter. This is useful if you need different configurations or want to test a new configuration.

Table 4.2 Global configuration options

Section	Name	Value
general	name	Name of the skillbed configuration (arbitrary string).

<table>
<tr><th>Section</th><th>Name</th><th>Value</th></tr>
<tr><td></td><td></td><td>Default: Skillbed</td></tr>
<tr><td></td><td>binary</td><td>Name of the binary to be tested.

Binaries are either be obtained in a collection of files, when this name is used to select the right file, or different versions of binaries come with different names, when this binary name is used to rename these different versions.

Default: model.exe</td></tr>
<tr><td></td><td>params</td><td>The default command line parameters to be used when starting a test. This value can be overwritten using the test configurations.</td></tr>
<tr><td></td><td>call</td><td>The default command line structure to be used when starting a test run. This value can be overwritten using the test configurations.

The following markers can be used:
<table>
<tr><td>{runid}</td><td>Unique skillbed run identifier</td></tr>
<tr><td>{binary}</td><td>Filename of binary</td></tr>
<tr><td>{type}</td><td>Type of run (custom or default)</td></tr>
<tr><td>{test}</td><td>Name of test</td></tr>
<tr><td>{run}</td><td>Name of run</td></tr>
<tr><td>{nodes}</td><td>Number of nodes</td></tr>
<tr><td>{runtime}</td><td>Maximum runtime in minutes</td></tr>
<tr><td>{exe_path}</td><td>Path to binary (incl. filename)</td></tr>
<tr><td>{bin_path}</td><td>Path to binary (excl. filename)</td></tr>
<tr><td>{input_path}</td><td>Path to input files of test</td></tr>
<tr><td>{test_path}</td><td>Path to working directory of test</td></tr>
<tr><td>{run_path}</td><td>Path to working directory of run</td></tr>
<tr><td>{params}</td><td>Command line parameter list</td></tr>
<tr><td>{params.X}</td><td>Single command line parameter (where X is the parameter index)</td></tr>
</table>
Default: {exe_path} {params}</td></tr>
<tr><td></td><td>build</td><td>The command line structure for compilation of the binary using Visual Studio.

The following markers can be used:
<table>
<tr><td>{exe_path}</td><td>Path to devenv.exe</td></tr>
<tr><td>{slnfile}</td><td>Path to solution file</td></tr>
<tr><td>{cf}</td><td>Configuration name</td></tr>
<tr><td>{pf}</td><td>Platform name</td></tr>
</table>
Default: "{exe_path}" {slnfile} /Rebuild "{cf}|{pf}"</td></tr>
<tr><td>storage</td><td>path</td><td>Path to central storage location.</td></tr>
<tr><td></td><td>scp</td><td>Path to the SCP storage location.

Syntax:</td></tr>
</table>

Section	Name	Value
		`scp = <webaddress>:<path_from_root>`
network	<any drive letter>	Network address, username and password separated by a space. The network address is mapped to the drive letter during initialization of the skillbed. For example: `p = \\filer\project user password`
servers	svn	URL to the Subversion repository where the skillbed configuration is hosted. Default: https://svn.oss.deltares.nl/repos/openearthtools/…/skillbed/
	hosts	Comma-separated list of hostnames of skillbed servers. Syntax: `hosts = <hostname>:<port>`
repositories	<any name>	Subversion repository address that can be used to download the latest model source code. The repository is referred to from the *binaries* section by its configuration name.
binaries	<any name>	Binary configuration settings. Two types exist: 1) Compilation using Visual Studio. This type consists of 5 values separated by spaces: i. Name of subversion repository (reference to *repositories* section) ii. Name of solution file including relative path from repository root iii. Configuration name iv. Platform name v. Name of resulting executable including relative path from repository root 2) Download. This type consist of 2 values separated by spaces: i. URL to executable or ZIP file containing the executable and possibly other files. URL's should start with the protocol (e.g. *http://*). For local files, use the file:// protocol followed by the drive letter and a colon (e.g. *file://d:/some_path/*) ii. Platform name
reports	<any name>	Report configuration settings. A report is configured using two values separated by spaces: i. Path to the main TEX file relative to the *latex* directory (e.g. *main/report.tex*) ii. Working directory for generating the reports relative to the *analysis* subdirectory in the run directory (e.g. *runs/<runid>*). All references to figures and tables are relative to this working directory. The working directory consists of the following components (in this order, but may be

<table>
<tr><th>Section</th><th>Name</th><th>Value</th></tr>
<tr><td></td><td></td><td>shortened): <<i>binary</i>>/<<i>type</i>>/<<i>test</i>>/<<i>run</i>>/. Where <<i>binary</i>> refers to an item in the <i>binary</i> section, <<i>type</i>> is either <i>default</i> or <i>custom</i> and <<i>test</i>> and <<i>run</i>> refer to the full test and run name respectively.

Default:
<code>main_custom = main/report.tex /trunk/custom/</code>

<code>main_default = main/report.tex /trunk/default/</code>

<code>cartoon = cartoon/report.tex /</code></td></tr>
<tr><td>templates</td><td>matlab</td><td>Relative path to main Matlab template. The Matlab script generated using this template is run to analyze the model results. For each test run, a rendered version of the matlab_test template is nested in this main template. This main template mainly handles the inclusion of toolboxes and global parameters.

The following markers can be used:
<table>
<tr><td>{funcname}</td><td>File/function name of Matlab script to be generated (temporary file name)</td></tr>
<tr><td>{matlabpath}</td><td>Path to additional Matlab functions (e.g. tools/matlab)</td></tr>
<tr><td>{networkpath}</td><td>Path to central storage location</td></tr>
<tr><td>{analysis_calls}</td><td>Nested code for individual test run analyses</td></tr>
</table>
Default: tools/templates/matlab/analysis.m.tmpl</td></tr>
<tr><td></td><td>matlab_test</td><td>Relative path to Matlab template for an individual test run. For each test run this template is rendered and nested in the main Matlab template defined by the matlab option.

The following markers can be used:
<table>
<tr><td>{runpath}</td><td>Path to test run directory with model output</td></tr>
<tr><td>{revision}</td><td>Revision number of executable being tested, if available. Otherwise it is zero.</td></tr>
<tr><td>{binary}</td><td>Name of binary (reference to the binaries section)</td></tr>
<tr><td>{type}</td><td>Type of run (custom or default)</td></tr>
<tr><td>{test}</td><td>Full test name</td></tr>
<tr><td>{run}</td><td>Full run name</td></tr>
<tr><td>{outputpath}</td><td>Path to analysis directory of current test run where output should be written</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Section</th><th>Name</th><th>Value</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><td>{datapath}</td><td>Path to data files for current test or test run directory (subdirectory of data directory)</td></tr>
<tr><td>{analysispaths}</td><td>List of subdirectories of the analysis directory that (partially) matches the full test and/or run name (e.g. subdirectories test_1, test_1_abs, test_1_abs.run_1 all matches the test run test_1_abs.run_1)</td></tr>
<tr><td>{analysisfunc}</td><td>Function name to be used for analysis of current test run</td></tr>
</table>

Default: tools/templates/matlab/analysis_item.m.tmpl</td></tr>
<tr><td></td><td>publish_bin</td><td>Relative path to HTML template to list published binaries. The only marker that can be used is the ${list} marker (mind the $), which will be replaced by an unordered HTML list and links to the latest binaries.

Default: tools/templates/publish/index_bin.html.tmpl</td></tr>
<tr><td></td><td>publish_report</td><td>Similar to the publish_bin option, but is used for generated reports instead of compiled binaries.

Default: tools/templates/publish/index_report.html.tmpl</td></tr>
<tr><td>mail</td><td>smtp</td><td>SMTP server address including port.</td></tr>
<tr><td></td><td>from</td><td>E-mailaddress used for from field.</td></tr>
<tr><td></td><td>subject</td><td>Subject of message to end-users containing the latest skillbed reports.</td></tr>
<tr><td></td><td>message</td><td>Path to file containing the message body relative to the tools directory. Default: mail.txt</td></tr>
<tr><td></td><td>notification</td><td>Path to file containing the message body for the notification e-mail relative to the tools directory.

The following markers can be used in this file:
<table>
<tr><td>{runid}</td><td>Unique skillbed run identifier</td></tr>
<tr><td>{storagepath}</td><td>Path to run results at the central storage location</td></tr>
</table>

Default: notification.txt</td></tr>
<tr><td>recipients</td><td><any e-mail address></td><td>List of e-mailaddress that should receive skillbed reports after a skillbed run is finished. The values are either all to send all available skillbed reports or a comma separated list with report names referring to the reports section.</td></tr>
</table>

4.1.1 Test lists configuration

As will be explained in the next section, it is possible to select which tests and runs to include in the current skillbed run using command line options. Possibly, you will need to run a large selection of tests over and over again. For these situations it is possible to store a specific selection of tests and runs in a file and use this file as command line argument. These files are stored in the *tools* directory.

Each line in a test list file contains a test to be included in the skillbed run. If only the full test name is given, all runs within that test are included. If you need to select specific runs, you can provide both the test and run name separated by a dot. To select two runs from a single test, you will need two lines.

For example:

```
test_1
test_2.run_1
test_2.run_3
test_4

...
```

4.1.2 Miscellaneous

There are some minor global configuration options, like the footer of the help text message. The help text message is showed upon request from the command line. The message has a footer containing contact details. This footer is stored as plain text in the *tools/footer.txt* file.

4.2 Test configuration

All test specific model input is stored in the *input* directory. This directory can contain an arbitrary structure of subdirectories. Such subdirectory, at an arbitrary level, is considered a test directory (i.e. a directory contain model input for a specific test) in case a file named *.config* (mind the dot) exists in that directory. This test directory may again contain an arbitrary subdirectory separating input for different runs from each other.

All directory names leading from the *input* directory to the *.config* file, concatenated by an underscore (_) define the full test name. The names of the runs are defined in the *.config* file regardless of the substructure in which the run data is found.

For example, two files define the configuration and model input of a test run: *input/project_1/test_1/.config* and *input/project_1/test_1/run_1/input.dat*. The full test name in this case is *project_1_test_1*. If the *.config* file defines a single run called *default*, the run name will be *default*, regardless of the fact that the model input is found in the subdirectory *run_1*.

The *.config* file has an INI structure that consists of several sections indicated with a header between brackets ([and]), like the global configuration file. Each section has a collection of name/value pairs that are explained in Table 4.3.

A run is defined as a separate section. The name of the section is *run_<name>*, where *<name>* is the run name. A separate section *categories* exists as well, but is not used by the skillbed, except for the categorization of tests in the skillbed reports. The categorization is defined in the *latex/overview.tex* file.

Table 4.3 Test configuration options

Section	Name	Value
general	enable	Enable (1) or disable (0) the test. Disabled tests are treated as non-existent.
	runs	Comma separated list of run names. Each run name should correspond to a section with the name *run_<name>*.
	responsible	Name and e-mailaddress (between < and >) of the person responsible for this test configuration.
	analysis	Matlab function name for analysis of test results. This

Section	Name	Value
		is the analysis of the combined result of all runs within this test and can co-exist with analyses for each individual run. The Matlab file should be stored in either the test or analysis directory.
run_<name>	enable	Enable (1) or disable (0) the run. Disabled runs are treated as non-existent.
	path	Path to the model input used for this run relative to the test directory.
	params	Command line parameters used for this run when executing the model executable.
	binaries	Comma separated list of binaries that should use this test (references to the *binaries* section in the global configuration). Set to *all* to enable the run for all binaries. Default: all
	types	Comma separated list of run types that should use this test (custom and/or default). Set to *all* to enable the run for all run types. Default: all
	platforms	Comma separated list of platforms that should use this test (win32 or unix). Set to *all* to enable the run for all platforms. Default: all
	nodes	Number of nodes to use for this run. A number larger than 1 will call the model using MPICH2. Default: 1
	analysis	Matlab function name for analysis of run results. The Matlab file should be stored in either the test, run or analysis directory.
	call	The command line structure to be used when starting a test run. The markers specified for the same option in the global configuration can be used.
	runtime	Maximum runtime in minutes. After this run period, the process will be killes. Default: 60
categories	<see *latex/overview.tex*>	

4.2.1 Analysis configuration

PLEASE BE PATIENT…

4.2.2 Report configuration

PLEASE BE PATIENT…

4.3 Configuration testing

PLEASE BE PATIENT…

5 Usage

5.1 Starting the skillbed

The skillbed is started from the command line. Both a server and client need to be started, in that order and possibly on different machines. It is also possible to run a server in local mode. In this case the server is started by a command line option of the client. Upon initialization of the client, a server is started on the same machine. Any communication of that skillbed run is restricted to that machine, so no communication with the central storage location or e-mailing will be performed.

Using command line options you can alter the settings used for a specific skillbed run. In principle, there are two types of command line options:

1) Flags, which are command line options that enable/disable a certain functionality

2) Name/value pairs, which give a certain option a specific value

Both types of options start with two dashes (--) followed by the option name. That's all for the flags. The name/value pairs are then followed by an equal sign (=) and the value. Some options come in an abbreviated syntax which consists of only a single dash (-) and a single character. The name/value pairs are then followed by a space (no equal sign!) and the value. Some name/value pairs accept a comma separated list of values. In these cases you may also specify the values *none* or *all* to select either no or all options. A full list of command line options can always we requested using the command line option --help or in abbreviated syntax -h.

All commands in this chapter are executed from the skillbed source directory, i.e. *tools/python/skillbed* and assume files with the *.py* extension to be associated with a Python installation with the necessary requirements (see 3.1 Requirements).

5.1.1 Server

The server is started by simply invoking the following command from the command line:

```
> server.py
```

By default, the server uses the configuration found in the *tools/skillbed.cfg* file and listens to port 8000. These default settings can be altered using the command line options in Table 5.1.

Table 5.1 Command line options for skillbed server/updater

Name	Default	Description
--port -p	8000	Port number to listen to.
--delay -d	0	Delay in seconds before starting server.
--config -c	tools/skillbed.cfg	Path to configuration file.
--help -h	-	Print help message.

The server registers itself at the central storage location for clients to find them. When using multiple servers on multiple machines, updating all servers with new code and tests may be a tedious job to do. Therefore an updater shell is created. The updater shell runs a server within

itself and functions like any other server. Upon request, however, the updater shell can update and reload the server. The updater shell is invoked as follows:

```
> updater.py
```

The updater shell accepts the same command line options as the server. The server itself is started by the updater shell and does not need to be started manually, in this case.

5.1.2 Client

Now you have either a skillbed server running, possibly within an updater shell, or you plan to run the skillbed in local mode and want to start the server upon client initialization. For now, lets assume the first case. Then the client is started by invoking the following command:

```
> client.py
```

This will initialize the client, but will not run any tests. Since no tests are ran, no analysis is performed, no reports are generated and no e-mails are sent. Therefore we need to specify a test, which can be done using command line parameters. For example:

```
> client.py –t all
```

In a sense, this command is the complement of the previous command, since it will run all tests and then run all analyses and generate all available reports and send these to all known users. So, be careful in your typing.

Different command line parameters exist for the client that either influence the workflow of the current skillbed run or are meant for maintenance of the skillbed system. All command line options for the client are listed in Table 5.2.

Table 5.2 Command line options for skillbed client

Name	Default	Description
--binaries -b	all	Comma separated list of binary names to be used (references to the *binaries* section of the global configuration).
--types -y	all	Comma separated list of type of runs to be used (custom and/or default).
--tests -t	none	Comma separated list of tests to be used.
--testlists -l	none	Comma separated list of files containing tests to be used (see 4.1.1 Test lists configuration)
--reports -r	all	Comma separated list of reports to be generated (references to the *reports* section of the global configuration).
--recipients -e	all	Comma separated list of registered e-mailaddresses. Each recipient will recieve an e-mail with the reports that are generated and are specified by the recipient as report of interest (references to the *recipients* section of the global configuration).
--config -c	tools/skillbed.cfg	Path to configuration file.
--servers -v		Comma separated list of servers to be used, including their port number (separated by a colon).
--source -o		Instead of compiling the source code obtained from the Subversion repository, you can specify a path to a local source code that needs to be compiled.

Name	Default	Description
--analyze -a		Instead of compiling/downloading the model and run all tests, use existing model results from a specific run with provided run id.
--skip-analysis	-	Skip running Matlab for the test analysis.
--skip-report	-	Skip generation of reports.
--skip-network	-	Skip copying results from and to network.
--skip-clean	-	Skip cleaning local results.
--skip-publish	-	Skip publishing of binaries and reports.
--test-report	-	Instead of starting a skillbed run, try to render one or more reports without content to see if the Latex code is correct.
--max-nodes -m	1000	Maximize the preparation of nodes. Can be used to quickly run a single run.
--notify -n		Send notification to this e-mailaddress when skillbed run is finished.
--interpret	-	Add interpretation of test results to reports (see 4.2.2 Report configuration).
--purge	-	Purge any run results and log files from client and servers. WARNING: This will actually delete a whole lot of data!
--gui -g	-	Start client with web GUI (see 5.2 Graphical User Interface (GUI)).
--service -s	-	Start client as web service. This is similar to the –gui option, but without starting a browser.
--local	-	Upon initialization of the client, start a server that listens to port 9000 (instead of 8000) and only use this server. This server will not be registered for the use with other clients.
--shutdown	-	Shutdown all servers afterwards. WARNING: This will shutdown all skillbed servers!
--update	-	Start client web GUI in debug mode. WARNING: This will cause huge security issues where your computer can be controlled remotely by anyone with a webbrowser.
--debug	-	Update skillbed servers before starting the skillbed run. This will include only the servers running in an updater shell (see 5.1.1 Server) WARNING: This will restart all skillbed servers!
--help -h	-	Print help message.

5.2 Graphical User Interface (GUI)

PLEASE BE PATIENT…

5.3 Update servers

PLEASE BE PATIENT…

5.4 Log files

PLEASE BE PATIENT…

6 Troubleshoot

6.1 Client

PLEASE BE PATIENT…

6.2 Server

PLEASE BE PATIENT…

7 Code structure

7.1 Packages

PLEASE BE PATIENT…

7.2 Workflow

PLEASE BE PATIENT…

7.3 Templates

PLEASE BE PATIENT…

7.4 Hooks

PLEASE BE PATIENT…