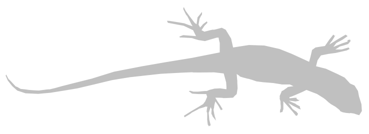
Eco-Hydrological Simulation Environment (echse)

Installation and Administration Guide



Author	David Kneis
Affiliation	Institute of Earth and Environmental Sciences Hydrology & Climatology Section, University of Potsdam, Germany
Contact	david.kneis [at] uni-potsdam.de

Project	PROGRESS
Sub-project	D2.2
Funding	German Ministry of Education and Research (BMBF)

Last update	February 17, 2014

Please help to improve this document by sending suggestions, corrections, wishes, and other useful feedback to the author (see above).

Contents

Chapter 1

Required external software

1.1 Before installing software

Be aware of the fact that the installation of any software, in particular if downloaded from the internet, is a potentially dangerous. If you install the suggested software, you do this at your own risk! To minimize the risk, it is recommended that you

- download the software from their official websites.
- scan downloaded files for viruses before execution.
- make regular backups of all important files to be prepared even for the worst case.

Note that you probably need administrator privileges to successfully install any of the software tools mentioned in the subsequent sections.

1.2 Statistical computing software ’R’

1.2.1 What is it good for?

Most of the `echse`-related pre- and post-processing tools are implemented in the R language. The `echse`’s code generator is currently implementend as an R-package too.

1.2.2 Basic installation

R is a widely used software for statistical computing and graphics creation (R Development Core Team, 2009). It is freely available for most platforms, including Linux and Windows. The official address of the project is `http://www.r-project.org`.

To find out whether R is installed on your system, type `R --version` at the command line. If R is available, a version info similar to the one shown below should appear.

```
dkneis@teufelsturm:/$ R --version
R version 2.10.1 (2009-12-14)
Copyright (C) 2009 The R Foundation [...]
ISBN 3-900051-07-0
[...]
dkneis@teufelsturm:/$
```

If this is not the case, you need to install R. Linux users need to use the package manager or an appropriate apt-command. Users of Windows need to go through the following steps:

1. Visit `http://www.r-project.org`
2. Save the installer for the latest version. The file name should be like `R-0.00.0-win.exe` (`0.00.0` = version).
3. Execute this file and follow the instructions.
4. Update your `PATH` environment variable to include the directory where the R binaries reside. If you installed R in folder `c:\program files`, for example, this will typically be `c:\program files\R\R-0.00.0\bin`. See Appendix A for information on how to set or modify environment variables.
5. Open a **new** terminal and check for a proper installation by querying the R version (see instructions above).

1.2.3 Installation of add-on packages

For some applications it may be necessary to install additional R-packages. These packages are not included in the basic R distribution and must be obtained separately.

Officially published R packages can be found on the CRAN website (CRAN: Comprehensive R Archive Network; see link at `http://www.r-project.org`). Typically, versions for several platforms and additional documentation material in PDF format is provided.

Other packages which are non officially published on CRAN are most often distributed as so-called tarballs. These are compressed archive files with the file extension `.tar.gz`.

On some systems (namely Windows), a menue is available at the top of the R console window that allows for convenient package installation, including download from CRAN. The system-independend way of package installation, however, uses the command line. Assuming that the package is available as a tarball, the following shell command should work on all platforms. In the example, 'pname' is the package name and 'x.y' is a version number.

```
R CMD INSTALL pname_x.y.tar.gz
```

To check whether an add-on package is properly installed use the R commands `library()` or `require()`.

For more help on package installation either type `?INSTALL` at the R prompt or use the shell command `R CMD INSTALL --help` from a normal terminal. Experience shows that package installation on Windows systems is sometimes difficult. If the installation fails due to permission issues, one could try to open a terminal with administrator privileges and then run the `R CMD INSTALL` command shown above.

Sometimes more difficulties arise if native source code (Fortran, C, C++) needs to be compiled during package installation. In such cases, the instructions given at `http://cran.r-project.org/doc/manuals/R-admin.html#The-Windows-toolset` may help. It seems that a common source of trouble is the existence of multiple parallel installations of a 'make' program or compilers (gfortran, gcc). In those cases, it may be necessary to modify the `PATH` environment variable (put the R-related directories first).

1.3 The GNU C++ compiler ’g++’

1.3.1 What is it good for?

g++ is the C++ compiler from the GNU compiler collection (gcc). It is a widely used, free software and runs on several platforms including Linux and Windows. It is used to compile the **echse**’s C++ source code. It may also be used to compile those pre- and/or post-processor tools which are implemented in C++.

1.3.2 Installation

To find out whether g++ is installed on your system, type `g++ --ver` at the command line. If the compiler is available, a version info similar to the one shown below should appear.

```
dkneis@teufelsturm:/$ g++ --ver
Using built-in specs.
Target: i486-linux-gnu
Configured with: ../src/configure -v     [...]
Thread model: posix
gcc version 4.4.3 (Ubuntu 4.4.3-4ubuntu5)
dkneis@teufelsturm:/$
```

If this is not the case, you need to install the g++ compiler. Linux users need to use the package manager or an appropriate apt-command. Users of Windows need to install MinGW and MSYS by going through the following steps:

1. Remove an older MinGW version if present. This is normally achieved by deleting the entire MinGW folder. You should make sure, however, that no other applications depend on that old version.
2. Visit `http://www.mingw.org`
3. From the download area, save the latest version of the installer. The file name should be like `mingw-get-inst-YYYYMMDD.exe` (`YYYYMMDD` = a date).
4. Execute this file. In the dialogue, select at least the following components for installation:
   - MinGW compiler suite – C++ compiler. This will install g++.
   - MinGW developer toolkit including the MSYS basic system.

   It is not a bad idea to also select the Fortran gfortran from the MinGW compiler suite for installation.
5. The user-selected path of the installation directory should not contain blanks (recommended: `c:/mingw`).
6. Note that the installer first downloads many files. Thus, you need an active internet connection.
7. Update your `PATH` environment variable to include the two directories
   - `c:\mingw\bin` (contains MinGW binaries) and
   - `c:\mingw\msys\<x.y>\bin` (contains MSYS binaries)

   where `c:\mingw` is the assumed (and recommended) installation directory and `<x.y>` is a version number. See Appendix A for information on how to set or modify environment variables.
8. Open a **new** terminal and check for a proper installation by querying the version of g++ (see instructions above).

1.4 The GNU Fortran compiler ’gfortran’

1.4.1 What is it good for?

gfortran is the Fortran compiler of the GNU compiler collection (gcc). It supports the Fortran 95 standard (at least), it is freely available and widely used. The gfortran compiler is required to compile Fortran code used by **`echse`**-related pre- and/or post-processing tools, namely some R packages.

1.4.2 Installation

To find out whether gfortran is installed on your system, type `gfortran --version` at the command line. If the compiler is available, a version info similar to the one shown below should appear.

```
dkneis@teufelsturm:/$ gfortran --version
GNU Fortran (Ubuntu 4.4.3-4ubuntu5) 4.4.3
Copyright (C) 2010 Free Software F[...]
[...]
dkneis@teufelsturm:/$
```

If this is not the case, you need to install gfortran. Linux users need to use the package manager or an appropriate apt-command. Users of Windows please go through all the steps described in the enumeration at the end of Sec. 1.3. In the dialog where you need to select components for installation, simply chose the Fortran compiler instead of (or in addition to) the C++ compiler.

1.5 MSYS (Windows users only)

MSYS provides basic Linux command line utilities for use on Windows systems. Using MSYS, it is possible to execute Linux bash shell scripts also on Windows systems. If you installed the C++ compiler g++ following the instructions in Sec. 1.3, you should already have MSYS on your system. If not, please follow the instructions in Sec. 1.3 and optionally skip the installation of a compiler.

If MSYS is properly installed, commands like `pwd` or `ls` should work when typed at a Windows command prompt. These commands show the current directory and list its contents, respectively.

1.6 A convenient text editor

On Linux systems, appropriate text editors are installed by default. On Windows, the default editor is hardly usable for serious text editing. NOTEPAD++ appears to be a good and free alternative.

Chapter 2

Installation of model engines

2.1 Introduction

This chapter deals with the installation of **`echse`**-based model engines from *existing* source code. The chapter does not cover the topic of model development or modification. The information is addressed to users of both Linux and Windows.

In this context, the term *model engine* is used to denote the binary file which is executed when performing a simulation. The term is introduced to allow for a clear distinction between the *model engine* (plain binary file) and the *model* (binary file + input data).

2.2 Software to install

2.2.1 Programs

You need to install the following programs in order to build an **`echse`**-based model engine:

- The 'R' software for statistical computing.
- The GNU C++ compiler 'g++'.
- Windows users: The bash-shell interpreter and commands provided by 'MSYS'.

Linux users probably want to use the package manager to install the software. Windows users should try the installers which can be downloaded from the respective websites. Please follow the installation instructions given in Chap. 1 as closely as possible. In particular, on Windows systems, it is important to add the binary directories of all the software to the `PATH` variable in order to make the programs accessible from any shell.

2.2.2 R-packages

You need to install the R-package **`codegen`** in order to run the **`echse`**'s code generator. You should find the latest version of this package in the sub-folder `R/packages` of the **`echse`**-tools main directory (see Sec. 2.3). The package is distributed as a tarball archive `codegen_x.y.tar.gz` where `x.y` is the current version number. See Sec. 1.2.3 for details on how to install R-packages.

2.3 The **echse** standard folders

2.3.1 Overview

At present, all **echse**-related files are split accross a small number of folders, also referred to as the **echse** *standard folders*. These folders are briefly explained in Table 2.1.

Table 2.1: *The* **echse** *standard folders.*

Folder	Contents
`echse_docs`	Documentations, mostly as LaTeX projects.
`echse_generic`	Common C++ source code being used by all **echse**-based model engines. Also contains generic scripts for engine building.
`echse_engines`	C++ source code of the classes being used by one or more particular model engines. This includes both generated and manually written code. Also contains text files with class and model declarations.
`echse_tools`	**echse**-related tools for pre- and post-processing of data.

It is assumed that you have obtained current versions of the **echse** standard folders listed in Table 2.1 either from repositories or by unpacking release archives. In order to install a model engine, you need at least the `echse_generic` and `echse_engines` folders whose contents is displayed in Figs. 2.1 & 2.2, respectively. Note that only the most important branches are displayed in these figures.

```
echse_generic --+-- core
                |
                +-- cpplib
                |
                +-- scripts
```

Figure 2.1: *Main branches of the* **echse** *standard folder with generic files.*

```
echse_engines --+-- bin
                |
                +-- classes
                |
                +-- def
                |
                +-- generated
                |
                +-- processes
```

Figure 2.2: *Main branches of the* **echse** *standard folder with class files for particular model engines.*

The functionality of many scripts and files contained in the folders `echse_generic` and `echse_engines` and its sub-folders depends on the integrity of the respective directory trees. Therefore, you should *not* rename, move, or delete any of the sub-folders and files without considering the potential consequences. Note that all references to files and directories are *relative*. Therefore, you can savely move each of the **echse** standard folders *as a whole* as long as you keep your system informed on their locations (see Sec. 2.4.1).

2.3.2 Definition of model engines

The following enumeration outlines the concept of model engines from a top-down point of view. It also gives a brief overview of the contents of the **echse** standard folder `echse_engines`.

1. Each model engine is composed of a particular set of classes. A rainfall-runoff model, for example, may comprise a sub-basin class, a reach class, and additional classes for special types of objects (reservoirs etc.). The set of classes being used in a particular model engines is listed in the text files in folder `echse_engines/def`.

2. Each of the classes is characterized by a particular set of data members (forcings, state variables, parameters, etc.). For each class, the declaration of data members can be found in the text files in folder `echse_engines/classes/declaration`. See Kneis (2012) for details.

3. Class declaration and implementation are well separated. The declaration part of the classes' source code is created by the code generator (output in `echse_engines/generated`). The implementation part consisting of manually written code to be found in `echse_engines/classes/implementation` (C++ include files).

4. The dynamics of state variables are due to the action of processes. In the **echse** concept, processes are represented by functions whose return value is typically a rate (mass or energy flux) or a derivative of a state variable with respect to time. Since a particular function may be used in multiple classes, the code is separated from the class implementation. It can be found in `echse_engines/processes`

2.4 Environment variables to be set

2.4.1 Pointers to the echse standard folders

The **echse** is not a software in the classical sense and, therefore, it does not require the typical installation procedure. There is neither an installer nor is the **echse** distributed as an installable software package. In particular, on a Windows system, the **echse** does not integrate into the so-called 'registry'.

Any **echse**-related files are contained in the standard folders introduced in Sec. 2.3. All you need to do is to inform the system on the location of these folders. For that purpose, you must define the environment variables listed in Table 2.2. Note that the variable names are UPPERCASE. See Appendix A and/or the help files of your operating system for detailed information on how to permanently set environment variables.

Table 2.2: *Environment variables pointing to the **echse** standard folders.*

Variable name	Value
`ECHSE_GENERIC`	Full path of the **echse** standard folder `echse_generic`.
`ECHSE_ENGINES`	Full path of the **echse** standard folder `echse_engines`.
`ECHSE_TOOLS`	Full path of the **echse** standard folder `echse_tools`.

On Windows systems, you need to use the forward slash (`/`) instead of the usual back-slash (`\`) to separate directory names in the value of the environment variables listed in Table 2.2. For example, if one of your **echse** standard folders is `d:\modeling\echse_generic`, the proper value of `ECHSE_GENERIC` would be `d:/modeling/echse_generic`. Disregard of this will cause problems during the compilation of model engines.

There must be no white space characters in the names of the **echse** standard folders. The existence of such characters (at any position) would break the functionality of some scripts. If you care for readability, use the _ character instead of white space, i. e. use `d:/my_files/echse_generic` instead of `d:/my files/echse_generic`, for example. Disregard of this will cause problems during the compilation of model engines.

If you ever move one (or all) of the **echse** standard folder(s), the variables listed in Table 2.2 need to be updated to reflect the new location(s). If you follow the recommendations in Sec. 2.4.2, no additional work is necessary.

2.4.2 Adjusting the PATH variable

Some sub-directories of the **echse** standard folders contain executable files (binaries and shell scripts). This is true in particular for the two folders `echse_engines/bin` (Fig. 2.2) and `echse_generic/scripts` (Fig. 2.1). It is recommended to add these two folders to the `PATH` environment variable. Only then you will be able to run **echse**-based model engines as well as code generation and build scripts from any terminal without typing full path names.

It is strongly recommended that you make use of the already-defined environment variables `ECHSE_ENGINES` and `ECHSE_GENERIC` when adding these folders to `PATH`. Then you can move the **echse** standard folders without updating `PATH` again. For example, Linux users should could add the following line to their shell initialization file (see also Appendix A):

```
export PATH=$PATH:$ECHSE_ENGINES/bin:$ECHSE_GENERIC/scripts
```

Windows users need to adjust the settings in an equivalent way using the control panel. For example, the variable `PATH` could have a contents similar to this one:

```
c:\mingw\bin;
c:\mingw\msys\1.0\bin;
c:\program files\R\R-2.15.2\bin;
%ECHSE_GENERIC%\scripts;
%ECHSE_ENGINES%\bin
```

Note that, to make this work, the `PATH` variable being set must be of the same category as the referenced variables `ECHSE_GENERIC` and `ECHSE_ENGINES`. This is due to the fact that a *user*-variable cannot be referenced in the definition of a *system*-variable and vice versa. Therefore, users without administrator privileges usually need to define a user-variable `PATH` in addition to the existing system-variable with that name.

2.5 Building a model engine

2.5.1 Pre-requisites

To be successsful, you should have read the sections Sections 2.3, 2.4, and 2.2. It is recommended that you perform some basic checks at a terminal prompt to make sure that the environment variable(s) are defined as intended and that the required software works properly.

2.5.2 Run the code generator

The code generator must be run from a shell prompt. On a Linux system, type the command `echse_generate` followed by a space and the name of the model engine for which code is to be generated.

Example:

```
david@falkenstein:~$ echse_generate myEngine
```

On a Windows system, use the command `echse_generate_win.bat` instead of `echse_generate`.

Example:

```
d:\> echse_generate_win.bat myEngine
```

To make the commands work at any prompt, the folder `echse_generic/scripts` (Fig. 2.1) must be part of the `PATH` variable (see Sec. 2.4.2).

The scripts provide some basic error checking. In most cases, you should be able to fix the problem yourself. Note that you can only generate code for model engines that have already been designed (by declaring the classes).

2.5.3 Run the build script

The build script must be run from a shell prompt. On a Linux system, type the command `echse_build` followed by a space and the name of the model engine. If want to skip the re-build of the static C++ library in order to speed up the compilation, you can set the second argument to 'n' (see example below). Use this option only if the library was just updated and note the warnings below.

Examples:

```
david@falkenstein:~$ echse_build myEngine
david@falkenstein:~$ echse_build myEngine n
```

On a Windows system, use the command `echse_build_win.bat` instead of `echse_build`.

Examples:

```
d:\> echse_build_win.bat myEngine
d:\> echse_build_win.bat myEngine n
```

To make the commands work at any prompt, the folder `echse_generic/scripts` (Fig. 2.1) must be part of the `PATH` variable (see Sec. 2.4.2).

The C++ library `libcpplib.a` is platform-specific, i. e. you cannot re-use the file when switching between operating systems and/or compiler versions. Disregard of this may yield an executable with subtile bugs which are very hard to trace. Thus, the library must always be re-build after a new installation of the **echse** software and each time the C++ compiler is upgraded. The re-build should only be skipped if you run a sequence of attempts to build a model engine and compile time really matters.

The build scripts provide some basic error checking. If yout get rather lengthy error messages, possible several pages, this indicates a compiler problem. Please make sure that (1) you have the complete source code for the model engine you try to build, (2) the code generator ran without reporting errors, (3) the C++ library was re-build, and (4) the C++ compiler is properly installed.

List of Figures

List of Tables

Bibliography

Kneis, D., 2012. Eco-Hydrological Simulation Environment (ECHSE) - Documentation of the Generic Components. University of Potsdam, Institute of Earth- and Environmental Sciences. URL: http://echse.bitbucket.org/downloads/documentation/echse_core_doc.pdf.

R Development Core Team, 2009. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL: http://www.R-project.org. ISBN 3-900051-07-0.

Appendix A

Setting environment variables

A.1 Linux

1. Open the shell initialization file in your home directory. On an Ubuntu system, this is

   ```
   /home/user_name/.bashrc
   ```

   Note that files whose names start with a dot are typically hidden.

2. To set a new variable, add a line like

   ```
   export myVar=value
   ```

   to the file. Don't use spaces before/after the '=' character.

3. Save the file.

To check your edits, open a **NEW** shell and try the following:

- Enter `echo $myVar` to display variable `myVar`.
- Alternatively, enter the `env` command to display all active variables.

In some situation it may be useful to update the `PATH` environment variable. The contents of this variable determines where the operation system searches for executable files. In order to add a path name to the `PATH` variable, append a line like `export PATH=$PATH:xxx` to the shell initialization file (see above). The colon is used to separate the added value (`xxx` in this example) from the existing ones.

Note that the value assigned to the `PATH` variable may contain references to other environment variables.

A.2 Windows

The following instructions should be applicable to most modern versions of Windows. The exact location of the dialog that allows for the manipulation of environment variables may vary, however.

1. - Windows XP: Click 'Start' → 'Control Panel', → 'Performance and Maintenance' → 'System'.
   - Windows 7: Click 'Start' → 'Control Panel', → 'System and Security' → 'System' → 'Advanced system settings'.

2. Go to the 'Advanced' tab and click 'Environment Variables'.

3. Locate the section with ’User variables’. These are the variables you can set/edit without administrator priviliges.
4. Click ’New’ to add a new variable (or ’Edit’ or ’Delete’ to modify/delete existing ones).

To check your edits, open a **NEW** shell and try the following:

- Enter echo \%myVar\% to display variable ’myVar’.
- Alternatively, enter the `set` command to display all active variables.

It is sometimes necessary to update the `PATH` environment variable. The contents of this variable determines where Windows searches for executable files. An update is usually required when installing software which does not use the Windows registry. In such cases, you typically need to add the name of the directory where the newly installed executable(s) reside to the existing value of the `PATH` variable. You best append the directory name at the very end of the using a semicolon as separator.

Consider the following example: If the `PATH` variable already contains the string `c:\program files\soft1` and you installed a software `xy` with the executable(s) in `f:\myPrograms`, the contents of your updated `PATH` variable should be `c:\program files\soft1; f:\myPrograms`.

Note that the value assigned to the `PATH` variable may contain references to other environment variables. This is restricted to variabes of the same category, however, i. e. you cannot mix system-variables and user-variables.