EC-CONF

User Manual

Table of Contents

Introduction

Software configuration can be a cumbersome task, particularly when it comes to complex systems with many components, different processing stages, and numerous files to be considered. Whenever more than a few configuration parameters, files and syntaxes are involved, or as soon as the configuration is to be done frequently, the process is error prone, too. Consistency and reproducibility are not easily achieved and documentation of a certain configuration is usually not included in the process.

The ec-conf tool is developed to aid the process of software configuration and to mitigate the problems just mentioned. As such, ec-conf is designed for the integration into the building and running cycle of a complex software system. Although ec-conf is being developed in the context of the coupled climate model EC-EARTH 3, it is not, per se, dependent on this software.

Configuration is needed, for example, when the software is built (i.e. compiled and linked) or when it is run in the context of a certain computing experiment. The configuration has to be redone whenever relevant aspects – such as the computing platform, user environment, or experiment settings – change.

Despite the fact the ec-conf's name bears a reference to EC-EARTH, it is best pronounced “easy conf”.

Design Principles

The fundamental principle behind ec-conf is that the process of software configuration implies the modification of configuration files according to configuration parameters. Configuration files are assumed to be text files and configuration parameters comprise a name and a value. The configuration parameter value is to be adapted according to the situation that the software is to be configured for.

That is, ec-conf creates configuration files by modifying configuration parameters according to their required values. This is achieved by reading the configuration parameters from an XML data base file and creating the configuration files with the help of templates. In order to control this process, ec-conf provides both a command line interface and a graphical user interface (GUI). Both interfaces can be used interchangeably, although they provide slightly different levels of control and comfort.

The main advantage of using ec-conf lies in the “one place, one syntax” principle, which implies that configuration is confined to just one file, where all configuration parameters are condensed, and, consequently, to one syntax. This principle holds regardless of the number of components and the number of configuration files and file types that are part of the software system.

Ec-conf has been implemented in Python, which should ensure widespread portability. Care has been taken to minimise the number of Python modules needed, although programming convenience has not been overly compromised. The ec-conf GUI makes use of Tkinter, however, the command line interface can be run even when Tkinter is not available.

Usage Principles

In order to utilise ec-conf, a user has to provide an XML data base file that contains all configuration parameters and a number of template files that are used to create

the configuration files. See later chapters about their syntax and other details. These files are usually created only once upon ec-conf's first use. Later on, only limited modifications are needed.

It is possible (and recommended) to introduce ec-conf gradually, which eases the transition from manual configuration. This is done by selecting just one configuration file as a template and start with few configuration parameters. Hence, only two files (one XML data base and one template) have to be provided for a start. In fact, any configuration file can be used as ec-conf template file without modification. It means that ec-conf is just passing the content of the template file without any changes. However, this implies that ec-conf has to be complemented by manual configuration.

When both the XML data base file and template file(s) are in place, the regular usage of ec-conf consists of two steps:

- Adapting the configuration parameter values in the XML data base file, and
- Running ec-conf in order to create the configuration files.

The first step is accomplished by editing the XML data base file (which is a text file) with a text editor of the user's choice. The second step means running either the command line or graphical user interface of ec-conf. Alternatively, both steps can be performed in one go with the graphical user interface.

Note that ec-conf only covers the configuration step! For example, if ec-conf is used for build configuration, the actual build commands (e.g. make) have to be launched after ec-conf has been run.

Command Line Interface

This chapter explains how to use the command line interface of ec-conf, while it is assumed that both the XML database file and the configuration template files are available. For details about how to create those files, see the later chapters. It is further assumed that the ec-conf script resides in the search path, although this is not strictly required.

Ec-conf recognises a number of command line options (switches), which can be specified either in long or short (one letter) form. To get basic help on the usage and options, run ec-conf with `--help` (`-h`):

```
> ec-conf --help

Usage:   ec-conf OPTIONS <XML_FILE>

Read configuration from an XML file and create configuration files.

ec-conf reads a data base of configuration parameters from an XML file.
Subsequently, a number of template files are processed in order to create
configuration files (targets).

Options: -h|--help                  Print this help text.
         -p|--platform <PLATFORM>   Selects the active platform, which has to
                                      be present among the platform sections
                                      defined in the XML file.
         -g|--gui                   Starts the graphical user interface. Turns
                                      off -x.
         -x|--write-xml             Writes the content of <XML_FILE> in XML
                                      format to stdout. This can be used to
                                      normalise the XML file and for tests.
```

```
        -v|--verbose                Produces verbose output (stderr). To
                                      increase verbosity, use more than once.
        -w|--no-warning             Turns off warnings (displays only errors).
```

The name of the XML data base file is an obligatory argument and this file is required to follow the XML syntax as explained in a later chapter.

Of the platform sections defined in the XML file, one platform can be selected as active with the `--platform` (`-p`) switch. Thus, all the configuration parameters defined in this platform section are used when the configuration files are created. It is possible to skip the platform switch, however, that means that no platform is active. Consequently, all configuration parameters defined within a platform section are ignored (and might be missing if referred to in a template file).

In order to start the graphical user interface (GUI) of ec-conf, the `--gui` (`-g`) option is used. Note that even in this case, the XML file has to be specified, and it is not possible to change the file from within the GUI (other than for writing out the contents into a new XML file).

The content of the XML data base file can be written to standard output (`stdout`) in the standard format defined by ec-conf. This is done with the `--write-xml` (`-x`) option for testing and debugging purposes, or in order to canonicalize the XML file. Note that error messages and warnings are written to `stderr`, such that the outputs can be separated.

Using the `--verbose` (`-v`) option on the command line makes the execution of ec-conf more verbose. The switch can be used several times to increase verbosity.

Finally, warnings can be suppressed during the execution by including the `--no-warning` (`-w`) option. However, error messages are still displayed.

XML Data Base File Syntax

The purpose of the XML data base file is to store all configuration parameters in a structured way. The eXtended Markup Language (XML) is built from a hierarchy of tags, i.e. a tag can contain a number of child tags. A tag may also include attributes and textual content.

A simplified description of the tag syntax reads

```
<TAG_NAME [ATTRIBUTE_NAME="ATTRIBUTE_VALUE" ...]>
    [CHILD_TAGS ...] or [TAG_CONTENT]
</TAG_NAME>
```

where `[]` denotes optional elements and `...` indicates repeating patterns. The `TAG_CONTENT` is either empty or made of one or more lines of text.

The XML data base file starts with the special line

```
<?xml version="1.0"?>
```

after which the `Configuration` tag (see description below) follows.

For a formal description of the XML data base file syntax, see the DTD (Document Type Definition) listing in the appendix. Hereafter follows a more verbal description of the tags that are needed to construct an ec-conf XML data base file.

Structural tags

The following tags are used to establish a hierarchy in the XML data base file. Thus, the configuration parameters are grouped by model components or platforms.

Configuration tag

Tag name	`Configuration`
Purpose	Mandatory root tag of the data base file, i.e. this tag is the beginning and end of the data base file. This tag envelops the entire collection of configuration parameters.
Attributes	None
Child tags	`Translation`, `Platform`, `Model`

Platform tag

Tag name	`Platform`
Purpose	Collects template/target pairs (translations) and parameters that are specific to a platform, i.e. allows translations and parameters to be grouped by platforms. None, one, or multiple platform tags are allowed.
Attributes	`Name`
Child tags	`Description`, `Translation`, `Parameter`

Model tag

Tag name	`Model`
Purpose	Collects configuration parameters that are specific to a (component) model, i.e. allows the parameters to be grouped by components. None, one, or multiple model tags are allowed.
Attributes	`Name`
Child tags	`Description`, `Parameter`

Content providing tags

The following tags provide the content of the XML data base file. This includes the actual configuration parameter values and auxiliary information.

Description tag

Tag name	`Description`
Purpose	Holds a detailed description about the parent tag in textual form. The description is intended to increase the readability of the XML data base file and is used by the `ec-conf` GUI. Users are encouraged to provide descriptions. This auxiliary tag is allowed once per parent tag.
Attributes	None
Child tags	None

Parameter tag

Tag name	`Parameter`
Purpose	Holds the name (as attribute) and the value (as child tag) of a configuration parameter. This is the place where actual configuration information is stored. Configuration parameters are associated with a type (see type tag below).
Attributes	Name
Child tags	`Description`, `Type`, `Value`

Type tag

Tag name	`Type`
Purpose	The type associated with a configuration parameter allows for automatic checking. Type can be one of ○ STRING ○ INTEGER ○ BOOLEAN ○ PATH ○ DATE **NOTE**: Automatic type checking is not yet implemented in ec-conf. However, it is recommended to use the type tag in order to increase the readability of the XML data base file and to enable future type checking.
Attributes	None
Child tags	None

Value tag

Tag name	`Value`
Purpose	The contents of this tag constitutes the actual configuration parameter value. None (for empty values), one, or multiple lines of text in the appropriate format, according to the type of the parameter, make up the content of the `value` tag.
Attributes	None
Child tags	None

Translation Tag

Tag name	`Translation`
Purpose	This tag specifies a pair of template and target files, which are named a translation in the context of ec-conf.
Attributes	`Name`
Child tags	`Description`, `Template`, `Target`, `Properties`

Template tag

Tag name	`Template`
Purpose	The name of the template file, including it's path is specified as tag content. Note that the path is either absolute (not recommended) or relative to the place from where ec-conf is executed.
Attributes	None
Child tags	None

Target tag

Tag name	`Target`
Purpose	The name of the target file, including it's path is specified as tag content. Note that the path is either absolute (not recommended) or relative to the place where ec-conf is executed.
Attributes	None
Child tags	None

Properties tag

Tag name	`Properties`
Purpose	Required file properties of the translations' target file are specified as tag content. This is used, for example, to ensure that the configuration file (target) is executable. Note that `executable` is the only supported file property for the time being.
Attributes	None
Child tags	None

Tag names

Note that the `Name` attributes of `Platform`, `Model`, and `Parameter` tags must correspond to the names used in the template files. That is, the `Name` attributes represent the link between the XML data base file and the template files. See Template Syntax described below.

Template File Syntax

Template files are used by ec-conf to create the actual configuration files (target files). Any text file can be used as a template file for ec-conf without modification. In fact, when the configuration process is modified to incorporate ec-conf, a gradual approach starting from unmodified configuration files is recommended. However, in many cases template files will already be present.

Basically, template files are just the configuration files of the software system, with the configuration parameter values marked by place holders. Hence, template files can be of any format (shell scripts, Fortran namelist files, Makefiles, ...), as long as they are text files.

Place-holder Syntax

In order to create a template file, place-holders are inserted at all positions where ec-conf is supposed to automatically substitute configuration parameter values. As an example, this snippet of a configuration file

```
ifs_numproc=1024
```

could be replaced by

```
ifs_numproc=[[[MOD:IFS:NUMPROC]]]
```

assuming that there is a model called `IFS` and a corresponding configuration parame-

ter called NUMPROC defined in the XML data base file.

The general form of a template file place holder is

```
[[[COMPONENT_TYPE:COMPONENT_NAME:PARAMETER_NAME]]]
```

where COMPONENT_TYPE is either PLT for platform or MOD for model, corresponding to the grouping tags allowed in the XML data base file, and PARAMETER_NAME is the name of a configuration parameter defined at the appropriate place in the same XML data base file. The whole term, including the brackets, is substituted by the parameter value defined in the XML file. No other changes than those confined by these bracket expressions are done to the template file.

Note that the substitution of the bracket patterns is done recursively. This allows references to other configuration parameters to be used as configuration parameter values within the XML data base file. However, this is probably an advanced feature not commonly used.

Active Platform

When writing platform dependent template files, a special platform named ACTIVE can be used in the substitution pattern:

```
[[[PLT:ACTIVE:PARAMETER_NAME]]]
```

This allows to refer to configuration parameters associated with the platform that the user indicates to be active at ec-conf run time. Selection of the active run time is either done using the --platform (-p) option on the ec-conf command line or through a GUI menu. Whatever platform is specified either way is then used for those substitution patterns that refer to the active platform.

A common usage example is to have, in a template file, a place-holder like

```
[[[PLT:ACTIVE:F90]]]
```

in order to select the Fortran90 compiler for whatever platform the user selects. This assumes that every platform section in the corresponding XML data base file has an F90 parameter defined.

Arithmetic Expressions

Sometimes, it is necessary to modify the parameter values before substitution. A common example may be a time period, which is given in hours in the XML data base file, that must be inserted in seconds in (some of) the configuration files. This can be achieved by applying a special syntax as seen in this example:

```
[[[MOD:IFS:run_length_hrs,3600,MUL]]]
```

This inserts the parameter run_length_hrs of the model named IFS after multiplying it by 3600, hence converting it from hours to seconds. As this example indicates, the operation must be noted in Polish Reverse Notation (RPN). In fact, more complex expressions involving multiple operands (both parameter names and numbers) and the following operations

Name	Operation	Example	Result
ADD	Addition	[[[1,2,3,ADD,ADD]]]	6
SUB	Subtraction	[[[6,2,SUB,4,SUB]]]	0

MUL	Multiplication	[[[2,10,MUL]]]	20
DIV	Division	[[[25,5,DIV]]]	5
POW	Exponentiation (Power)	[[[10,3,POW]]]	1000
MOD	Modulo (remainder after division)	[[[18,4,MOD]]]	2

(and any combinations thereof) can be used to compose the place-holders in template files.

A note to the advanced user: These computed substitution expressions are not limited to (integer and real) number computations. Due to ec-conf's implementation, all operations that result in a valid Python expression involving the above mentioned operations may be used. However, operands must be either parameter names or numbers.

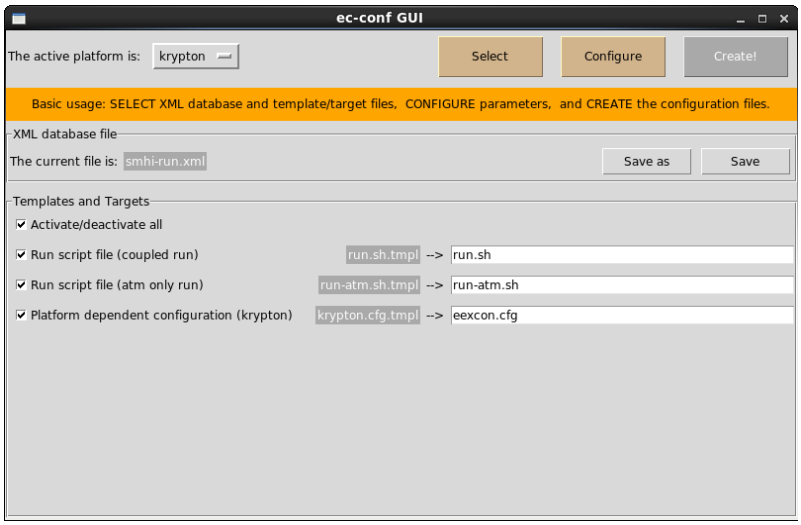
Graphical User Interface (GUI)

The ec-conf tool comes with a graphical user interface, which is started by including the `--gui` (`-g`) option on the command line:

```
> ec-conf --gui <OPTIONS> <XML_FILE>
```

Note that in this case, the `--write-xml` (`-x`) option does not work. Furthermore, it is important to note that informational messages, as well as warnings and errors are still written out to the command line (`stdout` and `stderr`) – even when using the GUI.

The GUI can be used instead of the command line interface in order to use all functions of ec-conf, in fact, the GUI is preferable for a more interactive way of working with ec-conf.



Selection panel of the ec-conf GUI

Once started, the ec-conf GUI shows up with the Selection panel, as seen in the figure to the left. On the upper right are three buttons, which control the main ec-conf functions. The **Select** button makes the Selection panel appear every time it is clicked. The next button, labelled **Configure**, shows the Configuration panel (described later) on every click. The rightmost button is named **Create!** and it starts the creation of the configuration files. These buttons are shown all the time when working with the ec-conf GUI.

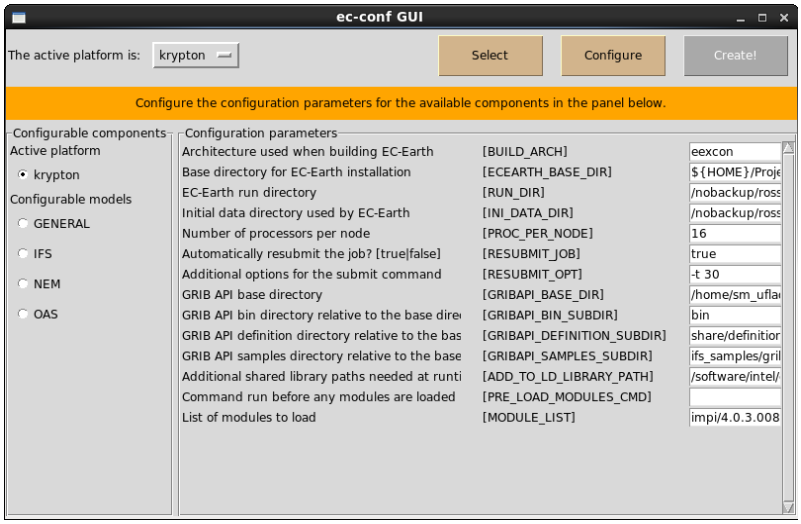
On the top left, the currently active platform is indicated. It is possible to change the active platform by choosing another entry from the option menu. The initial active platform is either taken from the `--platform` (`-p`) command line option or (if none was given) chosen by ec-conf.

A status line (in orange) is located below the active platform indicator. It will show

status messages, usage hints, and warnings when needed.

When the Selection panel is activated (at start-up or after clicking the **Select** button), the space below the status line is used to show and modify information about files. The current XML data base file is shown (at start-up time, as specified on the command line) and the **Save** button can be used to store the current values of the configuration parameters in that file. It is possible to save the data to a different file by using the **Save as** button, which will show a file dialogue window to specify a file path and name. This function can be used in order to not overwrite the current XML file or to create different XML files based on the same configuration parameters. A warning is shown when **Save as** is about to overwrite an existing file.

The bottom part of the Selection panel is holding information about the template and target files that are specified in the current XML data base file. They can individually or collectively be activated or deactivated by clicking the appropriate check buttons. Thus, it is possible to selectively create configuration files by clicking the **Create!** Button. Moreover, the file name and path of every configuration file (target) can be changed. This allows a very fine level of control over the configuration files that are created by ec-conf.



Configuration panel of the ec-conf GUI

When the **Configure** button is clicked, the GUI view changes to the Configuration panel (see figure to the left). This is the place where configuration parameter values are changed. The upper part of the window is just the same as in the Selection panel, with the **Select**, **Configure**, and **Create!** buttons and the active platform indicator. Likewise, the status line is taking up the space just below. Thereunder, information about the configuration parameters is displayed.

Since configuration parameters are grouped by platforms and component models, the left side is used to select the component from which configuration parameters are shown. When clicking on one of the components, the right hand side will adapt and show the corresponding configuration parameters, displaying their descriptions, names, and values. The values can be modified by changing the text entry fields provided for each parameters. The changes will take effect immediately, i.e. when the **Create!** button is pressed, the configuration files will be written according to the current parameter values. However, the XML data base files is only affected by the changes when the **Save** (or **Save as**) button on the Selection panel is used. Hence, it is important to save the changes to the parameter values if this is desired.

When the ec-conf GUI should be closed, the standard method of the underlying window manager for closing windows must be used, e.g. by pressing `Ctrl-Q` or `Alt-F4` or clicking the little cross at the window border.

Appendix: XML data base file DTD

```
<!ELEMENT Configuration (Translation*, Platform*, Model*)>

<!ELEMENT Translation (Description?, Template, Target, Properties?)>
<!ATTLIST Translation name CDATA #REQUIRED>

<!ELEMENT Platform (Description?, Translation*, Parameter*)>
<!ATTLIST Platform name CDATA #REQUIRED>

<!ELEMENT Model (Description?, Parameter*)>
<!ATTLIST Model name CDATA #REQUIRED>

<!ELEMENT Parameter (Description?, Type, Value)>
<!ATTLIST Parameter name CDATA #REQUIRED>

<!ELEMENT Description (#PCDATA)>
<!ELEMENT Template (#PCDATA)>
<!ELEMENT Target (#PCDATA)>
<!ELEMENT Properties (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```