# Ada Mode

An Emacs major mode for programming in Ada
Ada Mode Version 4.00

# Table of Contents

# 1 Overview

The Emacs mode for programming in Ada helps the user in understanding existing code and facilitates writing new code.

When the GNU Ada compiler GNAT is used, the cross-reference information output by the compiler is used to provide powerful code navigation (jump to definition, find all uses, etc.).

When you open a file with a file extension of `.ads` or `.adb`, Emacs will automatically load and activate Ada mode.

Ada mode works without any customization, if you are using the GNAT compiler (`https://libre2.adacore.com/`) and the GNAT default naming convention.

You must customize a few things if you are using a different compiler or file naming convention; See Section 3.2 [Other compiler], page 3, See Section 3.1 [Non-standard file names], page 3.

In addition, you may want to customize the indentation, capitalization, and other things; See Section 3.3 [Other customization], page 4.

Finally, for large Ada projects, you will want to set up an Emacs Ada mode project file for each project; See Chapter 5 [Project files], page 7. Note that these are different from the GNAT project files used by gnatmake and other GNAT commands.

See the Emacs info manual, section ’Running Debuggers Under Emacs’, for general information on debugging.

# 2 Installation

Ada mode is part of the standard Emacs distribution; if you use that, no files need to be installed.

Ada mode is also available as a separate distribution, from the Emacs Ada mode website `http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html`. The separate distribution may be more recent.

For installing the separate distribution, see the `README` file in the distribution.

To see what version of Ada mode you have installed, do *`M-x ada-mode-version`*.

The following files are provided with the Ada mode distribution:

- `ada-mode.el`: The main file for Ada mode, providing indentation, formatting of parameter lists, moving through code, comment handling and automatic casing.
- `ada-prj.el`: GUI editing of Ada mode project files, using Emacs widgets.
- `ada-stmt.el`: Ada statement templates.
- `ada-xref.el`: GNAT cross-references, completion of identifiers, and compilation. Also provides project files (which are not GNAT-specific).

# 3 Customizing Ada mode

Here we assume you are familiar with setting variables in Emacs, either thru 'customize' or in elisp (in your `.emacs` file). For a basic introduction to customize, elisp, and Emacs in general, see the tutorial in *The GNU Emacs Manual*.

These global Emacs settings are strongly recommended (put them in your .emacs):

```
(global-font-lock-mode t)
(transient-mark-mode t)
```

'`(global-font-lock-mode t)`' turns on syntax highlighting for all buffers (it is off by default because it may be too slow for some machines).

'`(transient-mark-mode t)`' highlights selected text.

See the Emacs help for each of these variables for more information.

## 3.1 Non-standard file names

By default, Ada mode is configured to use the GNAT file naming convention, where file names are a simple modification of the Ada names, and the extension for specs and bodies are '`.ads`' and '`.adb`', respectively.

Ada mode uses the file extensions to allow moving from a package body to the corresponding spec and back.

Ada mode supports a list of alternative file extensions for specs and bodies.

For instance, if your spec and bodies files are called *unit*`_s.ada` and *unit*`_b.ada`, respectively, you can add the following to your `.emacs` file:

```
(ada-add-extensions "_s.ada" "_b.ada")
```

You can define additional extensions:

```
(ada-add-extensions ".ads" "_b.ada")
(ada-add-extensions ".ads" ".body")
```

This means that whenever Ada mode looks for the body for a file whose extension is `.ads`, it will take the first available file that ends with either `.adb`, `_b.ada` or `.body`.

Similarly, if Ada mode is looking for a spec, it will look for `.ads` or `_s.ada`.

If the filename is not derived from the Ada name following the GNAT convention, things are a little more complicated. You then need to rewrite the function `ada-make-filename-from-adaname`. Doing that is beyond the scope of this manual; see the current definitions in `ada-mode.el` and `ada-xref.el` for examples.

## 3.2 Other compiler

By default, Ada mode is configured to use the GNU Ada compiler GNAT.

To use a different Ada compiler, you must specify the command lines used to run that compiler, either in lisp variables or in Emacs Ada mode project files. See Section 5.3 [Project file variables], page 8, for the list of project variables, and the corresponding lisp variables.

## 3.3 Other customization

All user-settable Ada mode variables can be set via the menu '`Ada | Customize`'. Click on the '`Help`' button there for help on using customize.

To modify a specific variable, you can directly call the function `customize-variable`; just type *`M-x customize-variable RET variable-name RET`*).

Alternately, you can specify variable settings in the Emacs configuration file, `.emacs`. This file is coded in Emacs lisp, and the syntax to set a variable is the following:

```
(setq variable-name value)
```

# 4 Compiling Executing

Ada projects can be compiled, linked, and executed using commands on the Ada menu. All of these commands can be customized via a project file (see Chapter 5 [Project files], page 7), but the defaults are sufficient for using the GNAT compiler for simple projects (single files, or several files in a single directory).

Even when no project file is used, the GUI project editor (menu '`Ada | Project | Edit`') shows the settings of the various project file variables referenced here.

## 4.1 Compile commands

Here are the commands for building and using an Ada project, as listed in the Ada menu.

In multi-file projects, there must be one file that is the main program. That is given by the `main` project file variable; it defaults to the current file if not yet set, but is also set by the "set main and build" command.

`Check file`
: Compiles the current file in syntax check mode, by running `check_cmd` defined in the current project file. This typically runs faster than full compile mode, speeding up finding and fixing compilation errors.

  This sets `main` only if it has not been set yet.

`Compile file`
: Compiles the current file, by running `comp_cmd` from the current project file.

  This does not set `main`.

`Set main and Build`
: Sets `main` to the current file, then executes the Build command.

`Show main`
: Display `main` in the message buffer.

`Build`
: Compiles all obsolete units of the current `main`, and links `main`, by running `make_cmd` from the current project.

  This sets `main` only if it has not been set yet.

`Run`
: Executes the main program in a shell, displayed in a separate Emacs buffer. This runs `run_cmd` from the current project. The execution buffer allows for interactive input/output.

  To modify the run command, in particular to provide or change the command line arguments, type *`C-u`* before invoking the command.

  This command is not available for a cross-compilation toolchain.

It is important when using these commands to understand how `main` is used and changed.

Build runs 'gnatmake' on the main unit. During a typical edit/compile session, this is the only command you need to invoke, which is why it is bound to *`C-c C-c`*. It will compile all files needed by the main unit, and display compilation errors in any of them.

Note that Build can be invoked from any Ada buffer; typically you will be fixing errors in files other than the main, but you don't have to switch back to the main to invoke the compiler again.

Novices and students typically work on single-file Ada projects. In this case, *C-c C-m* will normally be the only command needed; it will build the current file, rather than the last-built main.

There are three ways to change `main`:

1. Invoke '`Ada | Set main and Build`', which sets `main` to the current file.
2. Invoke '`Ada | Project | Edit`', edit `main` and `main`, and click '`[save]`'
3. Invoke '`Ada | Project | Load`', and load a project file that specifies `main`

## 4.2 Compiler errors

The `Check file`, `Compile file`, and `Build` commands all place compilation errors in a separate buffer named `*compilation*`.

Each line in this buffer will become active: you can simply click on it with the middle button of the mouse, or move point to it and press `RET`. Emacs will then display the relevant source file and put point on the line and column where the error was found.

You can also press the *C-x `* key (`next-error`), and Emacs will jump to the first error. If you press that key again, it will move you to the second error, and so on.

Some error messages might also include references to other files. These references are also clickable in the same way, or put point after the line number and press `RET`.

# 5 Project files

An Emacs Ada mode project file specifies what directories hold sources for your project, and allows you to customize the compilation commands and other things on a per-project basis.

Note that Ada mode project files `*.adp` are different than GNAT compiler project files `*.gpr`. However, Emacs Ada mode can use a GNAT project file to specify the project directories. If no other customization is needed, a GNAT project file can be used without an Emacs Ada mode project file.

## 5.1 Project File Overview

Project files have a simple syntax; they may be edited directly. Each line specifies a project variable name and its value, separated by "=":

```
src_dir=/Projects/my_project/src_1
src_dir=/Projects/my_project/src_2
```

Some variables (like `src_dir`) are lists; multiple occurrences are concatenated.

There must be no space between the variable name and "=", and no trailing spaces.

Alternately, a GUI editor for project files is available (see Section 5.2 [GUI Editor], page 8). It uses Emacs widgets, similar to Emacs customize.

The GUI editor also provides a convenient way to view current project settings, if they have been modified using menu commands rather than by editing the project file.

After the first Ada mode build command is invoked, there is always a current project file, given by the lisp variable `ada-prj-default-project-file`. Currently, the only way to show the current project file is to invoke the GUI editor.

To find the project file the first time, Ada mode uses the following search algorithm:

- If `ada-prj-default-project-file` is set, use that.
- Otherwise, search for a file in the current directory with the same base name as the Ada file, but extension given by `ada-prj-file-extension` (default `".adp"`).
- If not found, search for `*.adp` in the current directory; if several are found, prompt the user to select one.
- If none are found, use `default.adp` in the current directory (even if it does not exist).

This algorithm always sets `ada-prj-default-project-file`, even when the file does not actually exist.

To change the project file before or after the first one is found, invoke '`Ada | Project | Load ...`'.

Or, in lisp, evaluate `(ada-set-default-project-file "/path/file.adp")`. This sets `ada-prj-default-project-file`, and reads the project file.

You can also specify a GNAT project file to '`Ada | Project | Load ...`' or `ada-set-default-project-file`. Emacs Ada mode checks the file extension; if it is `.gpr`, the file is treated as a GNAT project file. Any other extension is treated as an Emacs Ada mode project file.

## 5.2 GUI Editor

The project file editor is invoked with the menu 'Ada | Projects | Edit'.

Once in the buffer for editing the project file, you can save your modification using the '[save]' button at the bottom of the buffer, or the *C-x C-s* binding. To cancel your modifications, kill the buffer or click on the '[cancel]' button.

## 5.3 Project file variables

The following variables can be defined in a project file; some can also be defined in lisp variables.

To set a project variable that is a list, specify each element of the list on a separate line in the project file.

Any project variable can be referenced in other project variables, using a shell-like notation. For instance, if the variable comp_cmd contains ${comp_opt}, the value of the comp_opt variable will be substituted when comp_cmd is used.

In addition, process environment variables can be referenced using the same syntax, or the normal $var syntax.

Most project variables have defaults that can be changed by setting lisp variables; the table below identifies the lisp variable for each project variable. Lisp variables corresponding to project variables that are lists are lisp lists.

In general, project variables are evaluated when referenced in Emacs Ada mode commands. Relative file paths are expanded to absolute relative to ${build_dir}.

Here is the list of variables. In the default values, the current directory "." is the project file directory.

ada_project_path_sep [default: ":" or ";"]

Path separator for ADA_PROJECT_PATH. It defaults to the correct value for a native implementation of GNAT for the current operating system. The user must override this when using Windows native GNAT with Cygwin Emacs, and perhaps in other cases.

Lisp variable: ada-prj-ada-project-path-sep.

ada_project_path [default: ""]

A list of directories to search for GNAT project files.

If set, the ADA_PROJECT_PATH process environment variable is set to this value in the Emacs process when the Emacs Ada mode project is selected via menu 'Ada | Project | Load'.

For ada_project_path, relative file paths are expanded to absolute when the Emacs Ada project file is read, rather than when the project file is selected.

For example if the project file is in the directory /home/myproject, the environment variable GDS_ROOT is set to /home/shared, and the project file contains:

```
ada_project_path_sep=:
ada_project_path=$GDS_ROOT/makerules
ada_project_path=../opentoken
```

then as a result the environment variable `ADA_PROJECT_PATH` will be set to `"/home/shared/makerules:/home/opentoken/"`.

The default value is not the current value of this environment variable, because that will typically have been set by another project, and will therefore be incorrect for this project.

If you have the environment variable set correctly for all of your projects, you do not need to set this project variable.

`bind_opt` [default: `""`]

Holds user binder options; used in the default build commands.

Lisp variable: `ada-prj-default-bind-opt`.

`build_dir` [default: `"."`]

The compile commands will be issued in this directory.

`casing` [default: `("~/.emacs_case_exceptions")`]

List of files containing casing exceptions. See the help on `ada-case-exception-file` for more info.

Lisp variable: `ada-case-exception-file`.

`check_cmd` [default: `"${cross_prefix}gnatmake -u -c -gnatc ${gnatmake_opt} ${full_current} -cargs ${comp_opt}"`]

Command used to syntax check a single file. The name of the file is substituted for `full_current`.

Lisp variable: `ada-prj-default-check-cmd`

`comp_cmd` [default: `"${cross_prefix}gnatmake -u -c ${gnatmake_opt} ${full_current} -cargs ${comp_opt}"`]

Command used to compile a single file. The name of the file is substituted for `full_current`.

Lisp variable: `ada-prj-default-comp-cmd`.

`comp_opt` [default: `"-gnatq -gnatQ"`]

Holds user compiler options; used in the default compile commands. The default value tells gnatmake to generate library files for cross-referencing even when there are errors.

If source code for the project is in multiple directories, the appropriate compiler options must be added here. Section 6.3 [Set source search path], page 15, for examples of this. Alternately, GNAT project files may be used; Section 6.4 [Use GNAT project file], page 16.

Lisp variable: `ada-prj-default-comp-opt`.

`cross_prefix` [default: `""`]

Name of target machine in a cross-compilation environment. Used in default compile and build commands.

`debug_cmd` [default: `"${cross_prefix}gdb ${main}"`]

Command used to debug the application

Lisp variable: `ada-prj-default-debugger`.

`debug_post_cmd` [default: `""`]

Command executed after `debug_cmd`.

`debug_pre_cmd` [default: `"cd ${build_dir}"`]

Command executed before `debug_cmd`.

`gnatfind_opt` [default: `"-rf"`]

Holds user gnatfind options; used in the default find commands.

Lisp variable: `ada-prj-gnatfind-switches`.

`gnatmake_opt` [default: `"-g"`]

Holds user gnatmake options; used in the default build commands.

Lisp variable: `ada-prj-default-gnatmake-opt`.

`gpr_file` [default: `""`]

Specify GNAT project file.

If set, the source and object directories specified in the GNAT project file are appended to `src_dir` and `obj_dir`. This allows specifying Ada source directories with a GNAT project file, and other source directories with the Emacs project file.

In addition, `-P{gpr_file}` is added to the project variable `gnatmake_opt` whenever it is referenced. With the default project variables, this passes the project file to all gnatmake commands.

Lisp variable: `ada-prj-default-gpr-file`.

`link_opt` [default: `""`]

Holds user linker options; used in the default build commands.

Lisp variable: `ada-prj-default-link-opt`.

`main` [default: current file]

Specifies the name of the executable file for the project; used in the default build commands.

`make_cmd` [default: `"${cross_prefix}gnatmake -o ${main} ${main} ${gnatmake_opt} -cargs ${comp_opt} -bargs ${bind_opt} -largs ${link_opt}"`]

Command used to build the application.

Lisp variable: `ada-prj-default-make-cmd`.

`obj_dir` [default: `"."`]

A list of directories to search for library files. Ada mode searches this list for the ‘`.ali`’ files generated by GNAT that contain cross-reference information.

The compiler commands must place the ‘`.ali`’ files in one of these directories; the default commands do that.

`remote_machine` [default: `""`]

Name of the machine to log into before issuing the compile and build commands. If this variable is empty, the command will be run on the local machine.

`run_cmd` [default: `"./${main}"`]

Command used to run the application.

`src_dir` [default: `"."`]
: A list of directories to search for source files, both for compile commands and source navigation.

# 6 Compiling Examples

We present several small projects, and walk thru the process of compiling, linking, and running them.

The first example illustrates more Ada mode features than the others; you should work thru that example before doing the others.

All of these examples assume you are using GNAT.

The source for these examples is available on the Emacs Ada mode website mentioned in See Chapter 2 [Installation], page 2.

## 6.1 No project files

This example uses no project files.

First, create a directory `Example_1`, containing:

`hello.adb`:

```
with Ada.Text_IO;
procedure Hello
is begin
   Put_Line("Hello from hello.adb");
end Hello;
```

Yes, this is missing "use Ada.Text_IO;" - we want to demonstrate compiler error handling.

`hello_2.adb`:

```
with Hello_Pkg;
procedure Hello_2
is begin
   Hello_Pkg.Say_Hello;
end Hello_2;
```

This file has no errors.

`hello_pkg.ads`:

```
package Hello_Pkg is
   procedure Say_Hello;
end Hello_Pkg;
```

This file has no errors.

`hello_pkg.adb`:

```
with Ada.Text_IO;
package Hello_Pkg is
   procedure Say_Hello
   is begin
      Ada.Text_IO.Put_Line ("Hello from hello_pkg.adb");
   end Say_Hello;
end Hello_Pkg;
```

Yes, this is missing the keyword `body`; another compiler error example.

In buffer `hello.adb`, invoke ‘`Ada | Check file`’. You should get a `*compilation*` buffer containing something like (the directory paths will be different):

```
cd c:/Examples/Example_1/
gnatmake -u -c -gnatc -g c:/Examples/Example_1/hello.adb -cargs -gnatq -gnatQ
gcc -c -Ic:/Examples/Example_1/ -gnatc -g -gnatq -gnatQ -I- c:/Examples/Example_1/hello.adb█
hello.adb:4:04: "Put_Line" is not visible
hello.adb:4:04: non-visible declaration at a-textio.ads:264
hello.adb:4:04: non-visible declaration at a-textio.ads:260
gnatmake: "c:/Examples/Example_1/hello.adb" compilation error
```

If you have enabled font-lock, the lines with actual errors (starting with `hello.adb`) are highlighted, with the file name in red.

Now type *`C-x`* `` ` `` (on a PC keyboard, ‘ is next to `1`). Or you can click the middle mouse button on the first error line. The compilation buffer scrolls to put the first error on the top line, and point is put at the place of the error in the `hello.adb` buffer.

To fix the error, change the line to be

```
    Ada.Text_IO.Put_Line ("hello from hello.adb");
```

Now invoke ‘`Ada | Show main`’; this displays ‘`Ada mode main: hello`’.

Now (in buffer `hello.adb`), invoke ‘`Ada | Build`’. You are prompted to save the file (if you haven’t already). Then the compilation buffer is displayed again, containing:

```
cd c:/Examples/Example_1/
gnatmake -o hello hello -g -cargs -gnatq -gnatQ -bargs  -largs
gcc -c -g -gnatq -gnatQ hello.adb
gnatbind -x hello.ali
gnatlink hello.ali -o hello.exe -g
```

The compilation has succeeded without errors; `hello.exe` now exists in the same directory as `hello.adb`.

Now invoke ‘`Ada | Run`’. A `*run*` buffer is displayed, containing

```
Hello from hello.adb

Process run finished
```

That completes the first part of this example.

Now we will compile a multi-file project. Open the file `hello_2.adb`, and invoke ‘`Ada | Set main and Build`’. This finds an error in `hello_pkg.adb`:

```
cd c:/Examples/Example_1/
gnatmake -o hello_2 hello_2 -g -cargs -gnatq -gnatQ -bargs  -largs
gcc -c -g -gnatq -gnatQ hello_pkg.adb
hello_pkg.adb:2:08: keyword "body" expected here [see file name]
gnatmake: "hello_pkg.adb" compilation error
```

This demonstrates that gnatmake finds the files needed by the main program. However, it cannot find files in a different directory, unless you use an Emacs Ada mode project file to specify the other directories; See Section 6.3 [Set source search path], page 15, or a GNAT project file; Section 6.4 [Use GNAT project file], page 16.

Invoke ‘`Ada | Show main`’; this displays `Ada mode main: hello_2`.

Move to the error with *`C-x`* `` ` ``, and fix the error by adding `body`:

```
package body Hello_Pkg is
```

Now, while still in hello_pkg.adb, invoke 'Ada | Build'. gnatmake successfully builds hello_2. This demonstrates that Emacs has remembered the main file, in the project variable main, and used it for the Build command.

Finally, again while in hello_pkg.adb, invoke 'Ada | Run'. The *run* buffer displays Hello from hello_pkg.adb.

One final point. If you switch back to buffer hello.adb, and invoke 'Ada | Run', hello_2.exe will be run. That is because main is still set to hello_2, as you can see when you invoke 'Ada | Project | Edit'.

There are three ways to change main:

1. Invoke 'Ada | Set main and Build', which sets main to the current file.
2. Invoke 'Ada | Project | Edit', edit main, and click '[save]'
3. Invoke 'Ada | Project | Load', and load a project file that specifies main

## 6.2 Set compiler options

This example illustrates using an Emacs Ada mode project file to set a compiler option.

If you have files from Example_1 open in Emacs, you should close them so you don't get confused. Use menu 'File | Close (current buffer)'.

In directory Example_2, create these files:

hello.adb:

```
with Ada.Text_IO;
procedure Hello
is begin
   Put_Line("Hello from hello.adb");
end Hello;
```

This is the same as hello.adb from Example_1. It has two errors; missing "use Ada.Text_IO;", and no space between Put_Line and its argument list.

hello.adp:

```
comp_opt=-gnatyt
```

This tells the GNAT compiler to check for token spacing; in particular, there must be a space preceding a parenthesis.

In buffer hello.adb, invoke 'Ada | Project | Load...', and select Example_2/hello.adp.

Then, again in buffer hello.adb, invoke 'Ada | Set main and Build'. You should get a *compilation* buffer containing something like (the directory paths will be different):

```
cd c:/Examples/Example_2/
gnatmake -o hello hello -g -cargs -gnatyt  -bargs  -largs
gcc -c -g -gnatyt hello.adb
hello.adb:4:04: "Put_Line" is not visible
hello.adb:4:04: non-visible declaration at a-textio.ads:264
hello.adb:4:04: non-visible declaration at a-textio.ads:260
hello.adb:4:12: (style) space required
gnatmake: "hello.adb" compilation error
```

Compare this to the compiler output in Section 6.1 [No project files], page 12; the gnatmake option `-cargs -gnatq -gnatQ` has been replaced by `-cargs -gnaty`, and an additional error is reported in `hello.adb` on line 4. This shows that `hello.adp` is being used to set the compiler options.

Fixing the error, linking and running the code proceed as in Section 6.1 [No project files], page 12.

## 6.3 Set source search path

In this example, we show how to deal with files in more than one directory. We start with the same code as in Section 6.1 [No project files], page 12; create those files (with the errors present)

Create the directory `Example_3`, containing:

`hello_pkg.ads`:

```
package Hello_Pkg is
   procedure Say_Hello;
end Hello_Pkg;
```

`hello_pkg.adb`:

```
with Ada.Text_IO;
package Hello_Pkg is
   procedure Say_Hello
   is begin
      Ada.Text_IO.Put_Line ("Hello from hello_pkg.adb");
   end Say_Hello;
end Hello_Pkg;
```

These are the same files from example 1; `hello_pkg.adb` has an error on line 2.

In addition, create a directory `Example_3/Other`, containing these files:

`Other/hello_3.adb`:

```
with Hello_Pkg;
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello_3
is begin
   Hello_Pkg.Say_Hello;
   Put_Line ("From hello_3");
end Hello_3;
```

There are no errors in this file.

`Other/other.adp`:

```
src_dir=..
comp_opt=-I..
```

Note that there must be no trailing spaces.

In buffer `hello_3.adb`, invoke ‘`Ada | Project | Load...`’, and select `Example_3/Other/other.adp`.

Then, again in hello_3.adb, invoke 'Ada | Set main and Build'. You should get a *compilation* buffer containing something like (the directory paths will be different):

```
cd c:/Examples/Example_3/Other/
gnatmake -o hello_3 hello_3 -g -cargs -I.. -bargs  -largs
gcc -c -g -I.. hello_3.adb
gcc -c -I./ -g -I.. -I- C:\Examples\Example_3\hello_pkg.adb
hello_pkg.adb:2:08: keyword "body" expected here [see file name]
gnatmake: "C:\Examples\Example_3\hello_pkg.adb" compilation error
```

Compare the -cargs option to the compiler output in Section 6.2 [Set compiler options], page 14; this shows that other.adp is being used to set the compiler options.

Move to the error with *C-x* `. Ada mode searches the list of directories given by src_dir for the file mentioned in the compiler error message.

Fixing the error, linking and running the code proceed as in Section 6.1 [No project files], page 12.

## 6.4 Use GNAT project file

In this example, we show how to use a GNAT project file, with no Ada mode project file.

Create the directory Example_4, containing:

hello_pkg.ads:

```
package Hello_Pkg is
   procedure Say_Hello;
end Hello_Pkg;
```

hello_pkg.adb:

```
with Ada.Text_IO;
package Hello_Pkg is
   procedure Say_Hello
   is begin
      Ada.Text_IO.Put_Line ("Hello from hello_pkg.adb");
   end Say_Hello;
end Hello_Pkg;
```

These are the same files from example 1; hello_pkg.adb has an error on line 2.

In addition, create a directory Example_4/Gnat_Project, containing these files:

Gnat_Project/hello_4.adb:

```
with Hello_Pkg;
with Ada.Text_IO; use Ada.Text_IO;
procedure Hello_4
is begin
   Hello_Pkg.Say_Hello;
   Put_Line ("From hello_4");
end Hello_4;
```

There are no errors in this file.

Gnat_Project/hello_4.gpr:

```
Project Hello_4 is
```

```
      for Source_Dirs use (".", "..");
   end Hello_4;
```

In buffer `hello_4.adb`, invoke '`Ada | Project | Load...`', and select `Example_4/Gnat_Project/hello_4.gpr`.

Then, again in `hello_4.adb`, invoke '`Ada | Set main and Build`'. You should get a `*compilation*` buffer containing something like (the directory paths will be different):

```
   cd c:/Examples/Example_4/Gnat_Project/
   gnatmake -o hello_4 hello_4 -Phello_4.gpr -cargs -gnatq -gnatQ -bargs  -largs
   gcc -c -g -gnatyt -gnatq -gnatQ -I- -gnatA c:\Examples\Example_4\Gnat_Project\hello_4.adb
   gcc -c -g -gnatyt -gnatq -gnatQ -I- -gnatA c:\Examples\Example_4\hello_pkg.adb
   hello_pkg.adb:2:08: keyword "body" expected here [see file name]
   gnatmake: "c:\examples\example_4\hello_pkg.adb" compilation error
```

Compare the `gcc` options to the compiler output in Section 6.2 [Set compiler options], page 14; this shows that `hello_4.gpr` is being used to set the compiler options.

Fixing the error, linking and running the code proceed as in Section 6.1 [No project files], page 12.

## 6.5 Use multiple GNAT project files

In this example, we show how to use multiple GNAT project files, specifying the GNAT project search path in an Ada mode project file.

Create the directory `Example_4` as specified in Section 6.4 [Use GNAT project file], page 16.

Create the directory `Example_5`, containing:

`hello_5.adb`:

```
   with Hello_Pkg;
   with Ada.Text_IO; use Ada.Text_IO;
   procedure Hello_5
   is begin
      Hello_Pkg.Say_Hello;
      Put_Line ("From hello_5");
   end Hello_5;
```

There are no errors in this file.

`hello_5.adp`:

```
   ada_project_path=../Example_4/Gnat_Project
   gpr_file=hello_5.gpr
```

`hello_5.gpr`:

```
   with "hello_4";
   Project Hello_5 is
      for Source_Dirs use (".");
      package Compiler is
         for Default_Switches ("Ada") use ("-g", "-gnatyt");
      end Compiler;
   end Hello_5;
```

In buffer `hello_5.adb`, invoke '`Ada | Project | Load...`', and select `Example_5/hello_5.adp`.

Then, again in `hello_5.adb`, invoke '`Ada | Set main and Build`'. You should get a `*compilation*` buffer containing something like (the directory paths will be different):

```
cd c:/Examples/Example_5/
gnatmake -o hello_5 hello_5 -Phello_5.gpr -g -cargs -gnatq -gnatQ -bargs  -largs
gcc -c -g -gnatyt -g -gnatq -gnatQ -I- -gnatA c:\Examples\Example_5\hello_5.adb
gcc -c -g -gnatyt -g -gnatq -gnatQ -I- -gnatA c:\Examples\Example_4\hello_pkg.adb
hello_pkg.adb:2:08: keyword "body" expected here [see file name]
gnatmake: "c:\examples\example_4\hello_pkg.adb" compilation error
```

Now type *`C-x`* `` ` ``. `Example_4/hello_pkg.adb` is shown, demonstrating that `hello_5.gpr` and `hello_4.gpr` are being used to set the compilation search path.

# 7 Moving Through Ada Code

There are several easy to use commands to navigate through Ada code. All these functions are available through the Ada menu, and you can also use the following key bindings or the command names. Some of these menu entries are available only if the GNAT compiler is used, since the implementation relies on the GNAT cross-referencing information.

*M-C-e* Move to the next function/procedure/task, which ever comes next (`ada-next-procedure`).

*M-C-a* Move to previous function/procedure/task (`ada-previous-procedure`).

*M-x ada-next-package*
Move to next package.

*M-x ada-previous-package*
Move to previous package.

*C-c C-a* Move to matching start of `end` (`ada-move-to-start`). If point is at the end of a subprogram, this command jumps to the corresponding `begin` if the user option `ada-move-to-declaration` is `nil` (default), otherwise it jumps to the subprogram declaration.

*C-c C-e* Move point to end of current block (`ada-move-to-end`).

*C-c o* Switch between corresponding spec and body file (`ff-find-other-file`). If point is in a subprogram, position point on the corresponding declaration or body in the other file.

*C-c c-d* Move from any reference to its declaration, for from a declaration to its body (for procedures, tasks, private and incomplete types).

*C-c C-r* Runs the `gnatfind` command to search for all references to the identifier surrounding point (`ada-find-references`). Use *C-x `* (`next-error`) to visit each reference (as for compilation errors).

If the `ada-xref-create-ali` variable is non-`nil`, Emacs will try to run GNAT for you whenever cross-reference information is needed, and is older than the current source file.

# 8 Identifier completion

Emacs and Ada mode provide two general ways for the completion of identifiers. This is an easy way to type faster: you just have to type the first few letters of an identifiers, and then loop through all the possible completions.

The first method is general for Emacs. It works by parsing all open files for possible completions.

For instance, if the words '`my_identifier`', '`my_subprogram`' are the only words starting with '`my`' in any of the opened files, then you will have this scenario:

```
You type:  myM-/
Emacs inserts:  'my_identifier'
If you press M-/ once again, Emacs replaces 'my_identifier' with
'my_subprogram'.
Pressing M-/ once more will bring you back to 'my_identifier'.
```

This is a very fast way to do completion, and the casing of words will also be respected.

The second method (*C-TAB*) is specific to Ada mode and the GNAT compiler. Emacs will search the cross-information for possible completions.

The main advantage is that this completion is more accurate: only existing identifier will be suggested.

On the other hand, this completion is a little bit slower and requires that you have compiled your file at least once since you created that identifier.

| | |
|---|---|
| *C-TAB* | Complete current identifier using cross-reference information. |
| *M-/* | Complete identifier using buffer information (not Ada-specific). |

# 9 Automatic Smart Indentation

Ada mode comes with a full set of rules for automatic indentation. You can also configure the indentation, via the following variables:

`ada-broken-indent` (default value: 2)
: Number of columns to indent the continuation of a broken line.

`ada-indent` (default value: 3)
: Number of columns for default indentation.

`ada-indent-record-rel-type` (default value: 3)
: Indentation for `record` relative to `type` or `use`.

`ada-indent-return` (default value: 0)
: Indentation for `return` relative to `function` (if `ada-indent-return` is greater than 0), or the open parenthesis (if `ada-indent-return` is negative or 0). Note that in the second case, when there is no open parenthesis, the indentation is done relative to `function` with the value of `ada-broken-indent`.

`ada-label-indent` (default value: -4)
: Number of columns to indent a label.

`ada-stmt-end-indent` (default value: 0)
: Number of columns to indent a statement `end` keyword on a separate line.

`ada-when-indent` (default value: 3)
: Indentation for `when` relative to `exception` or `case`.

`ada-indent-is-separate` (default value: t)
: Non-`nil` means indent `is separate` or `is abstract` if on a single line.

`ada-indent-to-open-paren` (default value: t)
: Non-`nil` means indent according to the innermost open parenthesis.

`ada-indent-after-return` (default value: t)
: Non-`nil` means that the current line will also be re-indented before inserting a newline, when you press `RET`.

Most of the time, the indentation will be automatic, i.e., when you press `RET`, the cursor will move to the correct column on the next line.

You can also indent single lines, or the current region, with `TAB`.

Another mode of indentation exists that helps you to set up your indentation scheme. If you press *`C-c TAB`*, Ada mode will do the following:

- Reindent the current line, as `TAB` would do.
- Temporarily move the cursor to a reference line, i.e., the line that was used to calculate the current indentation.
- Display in the message window the name of the variable that provided the offset for the indentation.

The exact indentation of the current line is the same as the one for the reference line, plus an offset given by the variable.

`TAB`
: Indent the current line or the current region.

*C-M-*
Indent lines in the current region.

*C-c TAB*
Indent the current line and display the name of the variable used for indentation.

# 10 Formatting Parameter Lists

*C-c C-f*    Format the parameter list (`ada-format-paramlist`).

This aligns the declarations on the colon (‘`:`’) separating argument names and argument types, and aligns the `in`, `out` and `in out` keywords.

# 11 Automatic Casing

Casing of identifiers, attributes and keywords is automatically performed while typing when the variable `ada-auto-case` is set. Every time you press a word separator, the previous word is automatically cased.

You can customize the automatic casing differently for keywords, attributes and identifiers. The relevant variables are the following: `ada-case-keyword`, `ada-case-attribute` and `ada-case-identifier`.

All these variables can have one of the following values:

`downcase-word`
: The word will be lowercase. For instance `My_vARIable` is converted to `my_variable`.

`upcase-word`
: The word will be uppercase. For instance `My_vARIable` is converted to `MY_VARIABLE`.

`ada-capitalize-word`
: The first letter and each letter following an underscore (‘`_`’) are uppercase, others are lowercase. For instance `My_vARIable` is converted to `My_Variable`.

`ada-loose-case-word`
: Characters after an underscore ‘`_`’ character are uppercase, others are not modified. For instance `My_vARIable` is converted to `My_VARIable`.

Ada mode allows you to define exceptions to these rules, in a file specified by the variable `ada-case-exception-file` (default `~/.emacs_case_exceptions`). Each line in this file specifies the casing of one word or word fragment. Comments may be included, separated from the word by a space.

If the word starts with an asterisk (‘`*`’), it defines the casing as a word fragment (or “substring”); part of a word between two underscores or word boundary.

For example:

```
DOD        Department of Defense
*IO
GNAT       The GNAT compiler from Ada Core Technologies
```

The word fragment `*IO` applies to any word containing “_io”; `Text_IO`, `Hardware_IO`, etc.

There are two ways to add new items to this file: you can simply edit it as you would edit any text file. Or you can position point on the word you want to add, and select menu ‘`Ada | Edit | Create Case Exception`’, or press *C-c C-y* (`ada-create-case-exception`). The word will automatically be added to the current list of exceptions and to the file.

To define a word fragment case exception, select the word fragment, then select menu ‘`Ada | Edit | Create Case Exception Substring`’.

It is sometimes useful to have multiple exception files around (for instance, one could be the standard Ada acronyms, the second some company specific exceptions, and the last one some project specific exceptions). If you set up the variable `ada-case-exception-file` as a list of files, each of them will be parsed and used in your emacs session. However, when you save a new exception through the menu, as described above, the new exception will be added to the first file in the list.

*C-c C-b* Adjust case in the whole buffer (`ada-adjust-case-buffer`).

*C-c C-y* Create a new entry in the exception dictionary, with the word under the cursor (`ada-create-case-exception`)

*C-c C-t* Rereads the exception dictionary from the file `ada-case-exception-file` (`ada-case-read-exceptions`).

# 12 Statement Templates

Templates are defined for most Ada statements, using the Emacs “skeleton” package. They can be inserted in the buffer using the following commands:

| | |
|---|---|
| *`C-c t b`* | exception Block (`ada-exception-block`). |
| *`C-c t c`* | case (`ada-case`). |
| *`C-c t d`* | declare Block (`ada-declare-block`). |
| *`C-c t e`* | else (`ada-else`). |
| *`C-c t f`* | for Loop (`ada-for-loop`). |
| *`C-c t h`* | Header (`ada-header`). |
| *`C-c t i`* | if (`ada-if`). |
| *`C-c t k`* | package Body (`ada-package-body`). |
| *`C-c t l`* | loop (`ada-loop`). |
| *`C-c p`* | subprogram body (`ada-subprogram-body`). |
| *`C-c t t`* | task Body (`ada-task-body`). |
| *`C-c t w`* | while Loop (`ada-while`). |
| *`C-c t u`* | use (`ada-use`). |
| *`C-c t x`* | exit (`ada-exit`). |
| *`C-c t C-a`* | array (`ada-array`). |
| *`C-c t C-e`* | elsif (`ada-elsif`). |
| *`C-c t C-f`* | function Spec (`ada-function-spec`). |
| *`C-c t C-k`* | package Spec (`ada-package-spec`). |
| *`C-c t C-p`* | procedure Spec (`ada-package-spec`. |
| *`C-c t C-r`* | record (`ada-record`). |
| *`C-c t C-s`* | subtype (`ada-subtype`). |
| *`C-c t C-t`* | task Spec (`ada-task-spec`). |
| *`C-c t C-u`* | with (`ada-with`). |
| *`C-c t C-v`* | private (`ada-private`). |
| *`C-c t C-w`* | when (`ada-when`). |
| *`C-c t C-x`* | exception (`ada-exception`). |
| *`C-c t C-y`* | type (`ada-type`). |

# 13 Comment Handling

By default, comment lines get indented like Ada code. There are a few additional functions to handle comments:

*M-;* Start a comment in default column.

*M-j* Continue comment on next line.

*C-c ;* Comment the selected region (add '`--`' at the beginning of lines).

*C-c :* Uncomment the selected region

*M-q* autofill the current comment.

# Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTEX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/licenses/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

# Index

(Index is nonexistent)