Guix Workflow Language Reference Manual

Reproducible Scientific Workflows based on Guix

The developers of the GNU Guix Workflow Language

Edition 0.2.0
16 February 2020

Table of Contents

Guix Workflow Language

This document describes the Guix Workflow Language version 0.2.0.

1 Introduction

This package provides the *Guix Workflow Language* (GWL), a scientific computing extension to the Guix package manager. It combines the specification of work units and their relationship to one another with the reproducible software deployment facilities of the functional package manager GNU Guix. A GWL workflow will always run in a reproducible environment that GNU Guix automatically prepares. The GWL extends your Guix installation with a single new sub-command: `guix workflow`.

In the GWL there are two concepts we need to know about: processes and workflows. We describe a computation (running a program, or evaluating a Scheme expression) using a process. A workflow describes how individual processes relate to each other (e.g. "process B must run after process A, and process C must run before process A").

GWL workflows are executable code. The workflow language is embedded in the powerful general purpose language Guile Scheme (`https://gnu.org/software/guile/`), so you can compute arbitrarily complex process and workflow definitions. The GWL supports a classic Lisp syntax as well as a Python-like syntax called Wisp (`https://www.draketo.de/light/english/wisp-lisp-indentation-preprocessor`).

2 Installation

There really is no point in using the GWL without Guix. If you already have a Guix installation, you can install the GWL with `guix install gwl`.

The Guix Workflow Language uses the GNU build system. To install it from a release tarball just unpack it and run the usual commands:

```
./configure
make
make install
```

If you want to build the sources from the source repository you need to bootstrap the build system first. Run `autoreconf -vif` first and then perform the above steps.

Note that in order for Guix to learn about the "workflow" sub-command provided by the GWL, the Guile module `(guix scripts workflow)` must be found in a directory on the `GUILE_LOAD_PATH`.

3 A Simple Workflow

To get a little taste of what the workflow language looks like, let's start by writing a simple workflow.

Here is a simple workflow example:

```
process greet
  packages "hello"
  # { hello }

process sleep
  packages "coreutils"
  # {
    echo "Sleeping..."
    sleep 10
  }

process eat (with something)
  name
    string-append "eat-" something
  # {
    echo "Eating {{something}}"
  }

process bye
  # { echo "Farewell, world!" }

workflow simple-wisp
  processes
    let
      :
        eat-fruit :  eat "fruit"
        eat-veges :  eat "vegetables"
      graph
        eat-fruit -> greet
        eat-veges -> greet
        sleep     -> eat-fruit eat-veges
        bye       -> sleep
```

This white-space sensitive syntax is called Wisp and if your familiar with Python or YAML you should feel right at home. To use this syntax simply save your workflow to a file ending on `.w`, `.wisp`, or `.gwl`.

The workflow language really is a *domain specific language* (DSL) embedded in Guile Scheme, so if you're a Lisper you may prefer to write your workflows directly in Scheme while basking in its parenthetical glow:

```
(import (gnu packages base))

(define-public greet
```

```
  (make-process
   (name "greet")
   (packages (list hello))
   (procedure '(system "hello"))))

(define-public sleep
  (make-process
   (name "sleep")
   (packages (list coreutils))
    (procedure
     '(begin
        (display "Sleeping...\n")
        (system "sleep 10")))))

(define-public (eat something)
  (make-process
   (name (string-append "eat-" something))
   (procedure
    `(format #t "Eating ~a\n" ,something))))

(define-public bye
  (make-process
   (name "bye")
   (procedure
    '(display "Farewell, world!\n"))))

(make-workflow
 (name "simple")
 (processes
  (let ((eat-fruit (eat "fruit"))
        (eat-veges (eat "vegetables")))
    (graph (eat-fruit -> greet)
           (eat-veges -> greet)
           (sleep     -> eat-fruit eat-veges)
           (bye       -> sleep)))))
```

Everything you can express in Scheme can also be expressed with the Wisp syntax, so the choice is down to personal preference.

4 Defining a Process

In the GWL a "process" is a combination of some kind of command or script to be executed, the software packages that need to be available when executing the commands, and declarations of inputs and generated outputs. A process has a name, and optionally a synopsis and a description, for display purposes.

We create a process with the `make-process` constructor like this:

```
make-process
  name "hello"
  procedure
     ' display "hello"
```

This creates a process with the name "hello", which will print the string `"hello"` once the process is executed. The `procedure` field holds the Scheme code that does all the work of saying "hello". We will talk about the `procedure` field a little later and show how to write code snippets in languages other than Scheme.

Often we will want to refer to previously created processes later, for example to combine them in a workflow definition. To do that we need to bind the created processes to variable names. Here we bind the above process to a variable named `hello`:

```
define hello
  make-process
    name "hello"
    procedure
       ' display "hello"
```

This is a very common thing to do, so the GWL offers a shorter syntax for not only creating a process but also binding it to a variable. The following example is equivalent to the above definition:

```
process hello
  procedure
     ' display "hello"
```

4.1 `process` Fields

Both `make-process` and `process` accept the same fields, which we describe below.

`name` The readable name of the process as a string. This is used for display purposes and to select processes by name. When the `process` constructor is used, the `name` field need not be provided explicitly.

`version` This field holds an arbitrary version string. This can be used to disambiguate between different implementations of a process when searching by name.

`synopsis` A short summary of what this process intends to accomplish.

`description`
A longer description about the purpose of this process.

`packages` This field is used to specify what software packages need to be available when executing the process. Packages can either be Guix package specifications —

such as the string `"guile@3.0"` for Guile version 3.0 — or package variable names. When using package variable names, you need to make sure to import the appropriate Guix module at the top of your workflow file, e.g. `(import (gnu packages guile))` for the variable `guile`.

The `packages` field accepts a list of packages as well as multiple values (an "implicit list"). All of the following specifications are valid. A single package:

```
process
  packages "guile"
  ...
```

More than one package:

```
process
  packages "guile" "python"
  ...
```

A single list of packages:

```
process
  packages
    list "guile" "python"
  ...
```

`inputs` This field holds inputs to the process. Commonly, this will be a list of file names that the process requires to be present. The GWL can automatically connect processes by matching up their declared inputs and outputs, so that processes generating certain outputs are executed before those that declare the same item as an input.

As with the `packages` field, the `inputs` field accepts an "implicit list" of multiple values as well as an explicit list. Additionally, individual inputs can be "tagged" or named by prefixing it with a keyword (see Section "Keywords" in *GNU Guile Reference Manual*). Here's an example of an implicit list of inputs spread across multiple lines where two inputs have been tagged:

```
process
  inputs
    . genome: "hg19.fa"
    . "cookie-recipes.txt"
    . samples: "foo.fq"
  ...
```

The leading period is Wisp syntax to continue the previous line. You can, of course, do without the periods, but this may look a little more cluttered:

```
process
  inputs genome: "hg19.fa" "cookie-recipes.txt" samples: "foo.fq"
  ...
```

Why tag inputs at all? Because you can reference them in other parts of your process definition without having to awkwardly traverse the whole list of inputs. Here is one way to select the first input that was tagged with the `samples:` keyword:

```
pick genome: inputs
```

To select the second item after the tag `genome:` do this:

```
pick second genome: inputs
```

or using a numerical zero-based index:

```
pick 1 genome: inputs
```

Chapter 5 [Code Snippets], page 12, for a convenient way to access named items in code snippets without having to define your picks beforehand.

`outputs` This field holds a list of outputs that are expected to appear after executing the process. Usually this will be a list of file names. Just like the `inputs` field, this field accepts a plain list, an implicit list of one or more values, and lists with named items.

The GWL can automatically connect processes by matching up their declared inputs and outputs, so that processes generating certain outputs are executed before those that declare the same item as an input.

`output-path`
This is a directory prefix for all outputs.

`run-time` This field is used to specify run-time resource estimates, such as the memory requirement of the process or the maximum time it should run. This is especially useful when submitting jobs to an HPC cluster scheduler such as Grid Engine, as these schedulers may give higher priority to jobs that declare a short run time.

`procedure`
This field holds an expression of code that should be run when the process is executed. This is the "work" that a process should perform. By default that's a quoted Scheme expression, but code snippets in other languages are also supported (see Chapter 5 [Code Snippets], page 12).

Here's an example of a process with a procedure that writes a haiku to a file:

```
process haiku
  outputs "haiku.txt"
  synopsis "Write a haiku to a file"
  description
    . "This process writes a haiku by Gary Hotham \
to the file \"haiku.txt\"."
  procedure
    ` with-output-to-file ,outputs
        lambda ()
          display "\
the library book
overdue?
slow falling snow"
```

The Scheme expression here is quasiquoted (with a leading `) to allow for unquoting (with ,) of variables, such as `outputs`.

Not always will Scheme be the best choice for a process procedure. Sometimes all you want to do is fire off a few shell commands. While this is, of course, possible to express in Scheme, it is admittedly somewhat verbose. For convenience we offer a simple and surprisingly short syntax for this common use case. As a bonus you can even leave off the field name "procedure" and write your code snippet right there. How? Chapter 5 [Code Snippets], page 12.

4.2 Process templates

When defining many similar processes, it can be useful to parameterize a single process template. This can be accomplished by defining a procedure that takes any number of arguments and returns a parameterized process. Here's how to do this somewhat verbosely in plain Scheme:

```
(define (build-me-a-process thing)
  "Return a process that displays THING."
  (make-process
    (name (string-append "show-" thing))
    (procedure ‘(display ,thing))))

;; Now use this procedure to build concrete processes.
(define show-fruit
  (build-me-a-process "fruit"))
(define show-kitchen
  (build-me-a-process "kitchen"))
(define show-table
  (build-me-a-process "table"))
```

As this is a somewhat common thing to do in real workflows, the GWL provides simplified syntax to express the same concepts with a little less effort:

```
process build-me-a-process (with thing)
  name
    string-append "show-" thing
  procedure
    ‘ display ,thing

define show-fruit
  build-me-a-process "fruit"
define show-kitchen
  build-me-a-process "kitchen"
define show-table
  build-me-a-process "table"
```

The result is the same: you get a procedure `build-me-a-process` that you can use to define a number of similar processes. In the end you have the three processes `show-fruit`, `show-kitchen`, and `show-table`.

4.3 Useful procedures

The `(gwl utils)` module provides a number of useful helpers that are intended to simplify common tasks when defining processes. The helpers defined by this module are all available by default.

`on` *collection higher proc* [Scheme Procedure]

The `on` procedure is an alternative way to express the application of a higher order function to some collection. The only purpose of this procedure is to improve legibility when using Wisp syntax, as it allows one to avoid leading dots. The following two expressions are equivalent:

```
;; With "on"
on numbers map
   lambda (number)
     + number 10

;; Without "on"
map
  lambda (number)
    + number 10
  . samples
```

`expand` *file-name-part...* [Scheme Procedure]

This procedure returns a list of file names by combining any number of file name parts given as arguments. A file name part can either be a string or a list of strings. This is very useful when you need to generate a list of input or output file names.

```
(expand "/home/"
        '("me" "you" "them")
        "/data/"
        '("old" "new"))

=> '("/home/me/data/old"
     "/home/me/data/new"
     "/home/you/data/old"
     "/home/you/data/new"
     "/home/them/data/old"
     "/home/them/data/new")
```

`pick` [*n*] *key collection* [Scheme Procedure]

This procedure allows you to pick a named item from a *collection* by looking for the specified keyword *key*. Optionally, you can provide a selector procedure or index *n* as the first argument. Without a selector the first item matching the given *key* will be returned. When the selector is `*` all items following the *key* (up to the next tag) will be returned. If the selector is a number it is used as a zero-based index into the list of items following the *key*. If the selector is a procedure it is applied to the list of items following the *key*.

```
define collection
  list
    . "one"
    . "two"
    . "three"
    . mine: "four"
    . "five"
```

```
    . yours: "six"

pick mine: collection

; => "four"

pick * mine: collection

; => '("four" "five")

pick second mine: collection

; => "five"

pick 0 yours: collection

; => "six"
```

`load-workflow` *file* [Scheme Syntax]

This macro lets you load a workflow from the given *file*. The file must evaluate to a workflow value. This macro is useful for when you want to extend previously defined workflows. The argument *file* is expected to be a file name relative to the file invoking `load-workflow`.

5 Code Snippets

The Guix Workflow Language is embedded in Guile Scheme, so it makes sense to use Scheme to define the work that a process should perform. Sometimes it may be more convenient, though, to express the procedure in a different language, such as GNU R, Python, or maybe even in Bash.

The GWL provides special syntax for embedding code snippets. The special syntax is provided in the `(gwl sugar)` module, and is loaded by default. Here is an example of a process that runs an embedded Bash shell script:

```
process run-bash
  packages "bash"
  # bash { echo "hello from bash!" }
```

Notice how the "procedure" field name was not used here, because the code snippet came last. This cuts down on boilerplate.

Code snippets are introduced with `# interpreter {`, where `interpreter` is the command line for running an interpreter, such as `/bin/bash -c`. Code snippets must end with a closing brace, `}`.

Make sure that the package inputs include a package providing the interpreter. For convenience we provide the special interpreters `bash`, `R`, and `python`, so that you don't have to specify a more complicated command line. When no interpreter is provided the generic shell interpreter `/bin/sh` will be used:

```
process run-sh
  # { echo "hello from a shell!" }
```

Within code snippets a special syntax is supported for accessing variables. Any uninterrupted value enclosed in double braces is considered a reference to a variable, which may also be the name of other process fields. In the following example, the shell snippet refers to the `name` and `inputs` fields of the current process:

```
process run-bash
  packages "bash"
  inputs
    . "a"
    . "b"
    . "c"
  # bash {
    echo "The name of this process: {{name}}."
    echo "The data inputs are: {{inputs}}."
  }
```

You can even access named or tagged values in lists. In the following example, the shell snippet refers to only selected values of the `inputs` field of the current process:

```
process run-bash
  packages "bash"
  inputs
    . "a"
    . mine: "b"
```

```
    . "c"
    . yours: "d"
  # bash {
    echo "This is mine: {{inputs:mine}}, and this is yours: {{inputs:yours}}."
  }
```

As expected, this will output the following text when run:

```
This is mine: b, and this is yours: d.
```

You can also access tagged sub-lists with the :: accessor:

```
process frobnicate
  packages "frobnicator"
  inputs
    . genome: "hg19.fa"
    . samples: "a" "b" "c"
  outputs
    . "result"
  # {
    frobnicate -g {{inputs:genome}} --files {{inputs::samples}} > {{outputs}}
  }
```

This process will cause the following command to be executed:

```
frobnicate -g hg19.fa --files a b c > result
```

You can also access process meta data through environment variables. The following variables may be set:

- _GWL_PROCESS_NAME
- _GWL_PROCESS_SYNOPSIS
- _GWL_PROCESS_DESCRIPTION
- _GWL_PROCESS_INPUTS
- _GWL_PROCESS_OUTPUT_PATH
- _GWL_PROCESS_OUTPUTS
- _GWL_PROCESS_COMPLEXITY_TIME
- _GWL_PROCESS_COMPLEXITY_SPACE
- _GWL_PROCESS_COMPLEXITY_THREADS

6 Defining a Workflow

A workflow is a combination of processes that run in a certain order or simultaneously. You can specify the dependencies of processes manually or let the GWL figure it out by matching up the declared inputs and outputs of all processes.

A workflow definition will look something like this:

```
workflow do-stuff
  processes
    . this
    . that
    . something-else
```

This defines a workflow with the name "do-stuff", binds it to a variable `do-stuff`, and declares that it consists of the three processes `this`, `that`, and `something-else`. All of these processes will be run at the same time. This may not be what you want when the processes depend on each other.

If the processes all declare inputs and outputs, the GWL can connect the processes and ensure that only independent processes are run simultaneously. Use the `auto-connect` procedure on your processes:

```
workflow do-stuff
  processes
    auto-connect
      . this
      . that
      . something-else
```

You can also explicitly construct a graph of processes with the aptly named `graph` macro. The following workflow definition lets the process `combine` run after `generate-A` and `generate-B`, which will both run in parallel. The process `compress` will run after `combine`, and thus at the very end.

```
workflow frobnicate
  processes
    graph
      combine -> generate-A generate-B
      compress -> combine
```

7 Process Engines

Once you have defined a workflow, there are different ways to run the processes it consists of. The simplest way is to turn the workflow into a Guile script that sets up the desired environment and then executes the workflow processes on the current machine. This is what the `simple-engine` does.

The `grid-engine` is similar to the `simple-engine` in that it generates a shell script, with the difference that it also includes resource variable definitions for submission to a Grid Engine scheduling system. The resource variables are derived from the process `run-time` field.

8 Invoking `guix workflow`

The Guix Workflow Language extends your Guix installation with a new sub-command: `guix workflow`. Here are some of the options the command accepts:

`--input=`*`name`*`[=`*`file`*`]`
`-i `*`name`*`[=`*`file`*`]`

A workflow may have so-called free inputs, inputs that are not provided by any of the workflow's processes. By default, the GWL will pick files from the current working directory that match the names of free inputs. This option can be used to map a *file* with an arbitrary name to a free input in the workflow with the given *name*. This option can be provided more than once.

In the following example, the free input called `genome` is mapped to the file `/data/hg19.fa` before running the workflow defined in `analysis.w`:

```
guix workflow --input=genome=/data/hg19.fa --run=analysis.w
```

`--output=`*`location`*
`-o `*`location`*

This option currently has no effect.

`--engine=`*`engine`*
`-e `*`engine`*

Select the process engine *engine* as the target of the generated process scripts. See Chapter 7 [Process Engines], page 15.

`--prepare=`*`file`*
`-p `*`file`*

Generate the process scripts and build or download all dependencies, but do not run the workflow process scripts corresponding to the workflow defined in *file*.

`--run=`*`file`*
`-r `*`file`*

Generate the process scripts, build or download all dependencies, and then run the workflow process scripts corresponding to the workflow defined in *file*.

`--dry-run`
`-n`

Prepare the scripts and the environments but don't actually run the processes. Only show what commands would be run.

`--force`
`-f`

Execute all processes, even if their outputs may have been cached from previous runs.

`--container`
`-c`

Run each process inside of an isolated environment with file system virtualization and user namespaces. Only declared input files will be available at execution time, and only declared output files will be stored. This is a great option to use when you want to make sure that your processes only depend on state that you have declared. A downside is that generated output files cannot be written to the target directories directly but are copied from the container to the file system.

`--graph=`*`file`*
`-g `*`file`*

Load the workflow *file* and generate a graph in Dot-format.

`--web-interface`
`-w` The GWL includes a web interface. This option starts it.

9 Acknowledgments

Thanks to the following people who contributed to the Guix Workflow Language through bug reports, patches, or through insightful discussions:

- Ludovic Courtès `ludo@gnu.org`
- Simon Tournier
- Kyle Meyer `kyle@kyleam.com`

Also thanks to the people who reviewed this project for joining the GNU project.

- Mike Gerwitz `mtg@gnu.org`

Thank you.

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 `http://www.gnu.org/copyleft/`.

    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.

Concept Index

S

T

U

V

Programming Index