The Vala Guide

Debdut

Version 0.1

March 17, 2015

The original form of this book is LaTeX source code. Compiling this LaTeX source has the effect of generating a device-independent representation of the book, which can be converted to other formats and printed.

The LaTeX source for this book is available from

http://github.com/vamega/The-Vala-Guide

Contents

Chapter 1

Some Background

1.1 Why not C?

C is a low level language programming language that can be run on a variety of platforms. A low level programming langauge is a programming language that exposes the programmer to the actual hardware that the code will run on. C comes with the C standard library, which in theory allows programmers to write code that can be compiled on different operating systems, and different processor architectures to create an executable file that will run on the system. However in practice the C standard library is not truly portable across operating systems, and in order to fix this the Glib library was developed.

Glib is a library written in C that is designed to be cross platform, and used as an alternative to the C standard library. In addition to providing functionality like printing text to the screen, and reading from files, it also provides a type system that allows for the implementation of classes, interfaces and objects in C. Also C being a primitive language, makes it very hard to implement new programming concepts.

1.2 Why Vala?

Vala is a programming language that is designed for Glib developers. Using Glib Vala provides a lot of features found in other high level programming languages in a manner that is concise and easy to use. It does this without introducing a virtual machine of any sort, and thus can be compiled to very efficiently written code.

Vala operates by converting Vala code into C source code and header files. But it is almost as fast as C in practice. It uses Glib's GObject type system to provide the object oriented features of Vala. The syntax of Vala is similar to C#, or Java, but modified so as to work with the GObject type system.

Some of the features that Vala supports are:

- Interfaces
- Properties
- Signals
- Foreach loops
- Lambda expressions
- Type inference for local variables
- Generics
- Non-null types
- Assisted memory management
- Exception handling

Vala does not introduce any feature that cannot be accomplished using C with Glib, however Vala makes using these features much simpler. One of the most common criticisms of Glib is that it is very verbose, and has a lot of boilerplate code. Using Vala, programmers can skip writing all the verbose code that C with Glib requires.

Vala is designed to allow the use of C libraries, especially GObject-based libraries easily. All that is needed to use a library written in C with Vala is a file describing the API, these files usually have a .vapi extension. This is different from other languages that use C libraries, which require special code, called a binding. that glues together the functionality of the language, and the C library. The problem with this is that when a library changes, the binding would have to be rewritten in order to continue being used with the library, as a result of which the bindings would be slightly behind the official implementation.

Programmes written in C can also make use of Vala libraries written in Vala without any effort whatsoever. The vala compiler will automatically generate the header files required, which C programmes need to use the library. It is also possible to write bindings for Vala from other languages like python and Mono.

More information about Vala is available at

http://live.gnome.org/Vala/

Chapter 2

The Basics of Vala

Let's begin writing programmes using Vala. The following is a simple programme that prints the words Hello World to the screen. Enter this code into your favourite text editor and save it in a file called `helloworld.vala`. Vala is a case-sensitive language, so when entering this code make sure not to

```
/*
 * helloworld.vala
 *
 * This is a simple Vala Program.
 * It prints the words Hello World to the screen.
 */
void main(string[] args) {
    // The following line prints the words 'Hello ,World'.
    stdout.printf("Hello, World\n");
}
```

2.1 Compiling the code

To compile this programme, open up a terminal, change the directory to the directory where you saved your file, and type **valac** followed by the name of the source file, as shown here.

```
$valac helloworld.vala
```

If there were no errors then it will appear as if nothing has happened. However a new executable file named `helloworld` would have been created by the compiler. You can run this programme by typing it's name preceded by `./` as is shown below.

```
$./helloworld
```

You will see the following output.

```
Hello, World
```

2.2 Understanding the code

Despite it's length, the hello world programme introduces a number of features and concepts that are used in Vala. The programme starts with the following lines:

```
/*
 * helloworld.vala
 *
 * This is a simple Vala Program.
 * It prints the words Hello World to the screen.
 */
```

This is a comment. A comment in Vala is text that is found in the source code, that is ignored by the compiler. A comment is generally used to describe the operation of the programme, the reasoning behind some code, or an explanation of how some section of code works.

Vala uses two types of comments. The comment seen in the section above is called a *multi-line comment*. It is named so because the comment can span multiple lines in the source code. This type of comment begins with the characters /* and ends with the characters */, with everything between these characters being the comment.

```
void main(string[] args) {
```

This line defines the `main( )` method. The `main( )` method is the code that is executed when the programme is run. It can be thought of as the starting point of a programme's execution. All Vala (and for that matter, C, C++, C#, among others) applications begin by running the code in the `main( )` method.

When a programme is executed by the operating system, the operating system runs the code in the main method, and when the programme has finished execution, it may return a number to the operating system. This number could be used to inform the operating system whether the programme finished execution without any problems, or could be used to signal a problem that had occurred during the execution of the programme. The keyword `void` in this line simply tells the compiler that `main( )` will not return a value. All methods can return a value, as you will see later on.

Vala is a language which is case-sensitive. So `void` cannot be written as `VOID` or `Void`. The `main( )` method must be named exactly as is, as any change in capitalisation would result in the file generated being unexecutable. This is because the compiler would not see a `main( )` method, and would thus compile the code assuming that it code meant to be used in another programme.

The last character on the line is the `{`. This signals the start of `main( )`'s body. All of the code that comprises a method will occur between the method's opening curly brace and its closing curly brace.

One other point: `main( )` is simply a starting place for your programme. A complex programme will have dozens of classes, only one of which will need to have a `main( )` method to get things started.

```
// The following line prints the words 'Hello ,World'.
```

This line is a *single-line comment*, the other type of comment supported by Vala. A single-line comment commences with the character sequence `//` and terminates at the end of the line. They are generally used to add short explanations to particular lines of code, while multi-line comments are used to add longer explanations.

```
stdout.printf("Hello, World\n");
```

This line outputs the string of characters “Hello, World” to the screen, followed by a new-line character, that moves subsequent output to the next line. The `printf( )` method shown here is part of GLib a library that Vala relies upon extensively. The `printf` method will print the text inside the parenthesis to the screen. The `\n` is a special character sequence that represents a newline.

Notice that the `printf( )` statement ends with a semicolon. All statements in vala end with a semicolon. The only statement in this programme is the `printf( )` statement. All the other lines are not statements - they are either comments, method declarations, or in the case of lines with only the `{` or `}` characters, scope descriptors.

2.3 Programming with Variables

Although the programme written above instructs the computer, it is not very useful as everything in the programme was static. The programme would produce the same output on every run, and was not doing any calculation. Fundamental to making programmes more dynamic is the concept of a variable. A variable is just a name for a certain memory location. It can be thought of as a box where you can store data. The the data in a variable can be modified as a programme is running, thus allowing for a programme to become dynamic.

This next programme introduces a few basic types of variables that are available in Vala.

```
/*
 * variable_example.vala
 *
 * This programme demostrates how to use variables, and introduces a few
 * types of variables that are available in Vala.
 */
void main(string[] args) {
    // Declaring variables
    bool truth = false;
    unichar letter = 'v';
    int num = 10;
    double decimal_point = 15.33;
    string name = "Varun Madiath";

    // Printing the values in the variables.
    stdout.printf(truth.to_string() + "\n");
    stdout.printf(letter.to_string() + "\n");
    stdout.printf(num.to_string() + "\n");
    stdout.printf(decimal_point.to_string() + "\n");
    stdout.printf(name + "\n");

    // Modifying a variable.
    num = num * 2;

    // Printing out modified variable
    stdout.printf(num.to_string() + "\n");
}
```

You can compile and run this programme by typing

```
$ valac variable_example.vala
```

```
$ ./variable_example
```

The output will look like this.

```
false
v
10
15.33
Varun Madiath
20
```

Let's now dissect this code to understand how this output was generated.

```
bool truth = false;
unichar letter = 'v';
int num = 10;
double decimal_point = 15.33;
string name = "Varun Madiath";
```

These lines create variables, and assign them a value. There are different types of variables being created here. The `bool` variable represents a boolean, or a variable which has only two possible values, `true` and `false`. The unichar variable represents a character of text, so almost any character can be stored in this variable (yes that includes characters from Japanese and Indic scripts). The character is enclosed in single-quation marks to indicate that it is a character, and not an command for the computer. The `int` variable represents an integer (a number which doesn't have a decimal point in it). The double variable represents a floating point number (a variable which may or may not have a decimal point in it) and finally a `string` variable represents a series (or string) of characters. A string is enlosed in double quotation marks to demarcate the text as a string.

Variables are assigned values with the `=` operator, with the name and type of the operator on the left side of the operator, and the value on the right side. The general form of a variable declaration can thus be written as (text inside square brackets is optional).

```
type name [= value]
```

```
stdout.printf(truth.to_string() + "\n");
stdout.printf(letter.to_string() + "\n");
stdout.printf(num.to_string() + "\n");
stdout.printf(decimal_point.to_string() + "\n");
stdout.printf(name + "\n");
```

These lines print the values of the variables to the screen. Printing to the screen is done using the same `printf()` method that was used in the first example. You will notice that inside the `printf()` method most variable names are appended with `.to_string()`. This is because the `printf()` method must be given a string to print to the screen. This is also the reason why the string variable is not appended with `.to_string()`.

```
num = num * 2;
```

This line modifies the value of `num` by assigning it a new value. The new value is given by the expression `num * 2`. It is important to not look at this as an algebraic expression, as that would not make sense, but rather as an assignment. There is another operator for equality, that we will discuss later. Here the value of `num` is multiplied by two, and the result is stored in `num`.

```
stdout.printf(num.to_string() + "\n");
```

This line prints the value of `num`. As you can see in the output, the value has changed after the assignment above.

2.4 A runthrough of some basic Vala Grammar

We have now written two small programmes in Vala, but before we proceed any further it would be wise to have a look at the various elements that make a programme. The different elements that compose a Vala programme are:

- Whitespace
- Comments
- Identifiers
- Keywords
- Literals
- Operators
- Separators

2.5 Whitespace

Vala is a free-form language. A free-form language is one where code is not required to be formatted in any particular fashion. The programmes that we have written till now could be condensed into a single line (with the exception of the single line comments). The only requirement is that every token is separated by at least one character of whitespace, unless it is already separated by a separator or operator. Separators and operators will be discussed in just a moment.

2.6 Comments

We have already introduced the two types of comments that Vala supports. The single-line comment and the multi-line comment.

It is important to note that the sequence of characters */ cannot appear anywhere in your comment, as the compiler will understand that character sequence to be the end of the comment. This implies that one inline comment may not be placed inside another. As you begin to write longer, more complicated programmes, you might wish to comment out a section of your code in order to try tracking the source of an error. In such cases uses a multi-line comment, is inadvisable unless you specifically know that there is no multi-line comment in that block of code. It is preferrable to use your editor to prepend each line in that code block with `//` as this cannot cause any problems. Most code editors have a feature that allows you to do this.

2.7 Identifiers

Identifiers are used for method and variable names. In the programmes we have written until now we have always written a `main()` method, in these methods the identifier is `main`. All the variable names in the second example were identifiers. Less obvious identifiers were `stdout` and `printf()`, these are two different identifiers separated by a separator - the period. Identifiers in Vala must can only contain the characters [A-Z], [a-z], [0-9] or an underscore, but the first character may not be a digit.

2.8 Keywords

Vala has a set of words that cannot be used as identifiers. These words are reserved for expressing information about the programme to the compiler.

The list of reserved keywords is show in the following table:

if	else	switch	case	default	do	while
for	foreach	in	break	continue	return	try
catch	finally	throw	lock	class	interface	struct
enum	delegate	errordomain	const	weak	unowned	dynamic
abstract	virtual	override	signal	extern	static	async
inline	new	public	private	protected	internal	out
ref	throws	requires	ensures	namespace	using	as
is	in	new	delete	sizeof	typeof	this
base	null	true	false	get	set	construct
default	value	construct	void	var	yield	global
owned						

2.9 Literals

Literals are the constructs that represent values in the programme. An example of a constant here would be the value `true`, for a boolean, or `Varun` for a string. Here are some examples of literals:

- 1824
- 1858.58
- true
- false
- null
- ”Varun Madiath”

The first literal represents an integer, the next represents a floating-point value, the third and fourth are boolean constants, the fifth is the null constant, and the last represents a string. A literal can be substituted for a variable anywhere that a variable of its type is allowed. If you've actually read the entire list of keywords you would notice that the literals `true`, `false`, and `null` are also keywords.

2.10 Separators

In Vala, there are a few characters that are used as separators. The most commonly used separator in Vala is the semicolon. As you have seen, it is used to terminate statements. The separators are shown in the following table:

Symbol	Name	Purpose
( )	Parentheses	They are used to contain the list of parameters in method definition and invocation. They are also used to define precedence and contain expressions. Finally they are used to contain the data type when performing static type casting.
{ }	Curly Braces	Used to contain the values of automatically initialised arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a for statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

Appendix A

GNU Free Documentation License

Version 1.1, March 2000

Preamble

The purpose of this License is to make a manual, textbook, or other written 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.

A.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document," below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you."

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. (For example, 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.

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 "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and 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 has been designed to thwart or discourage subsequent modification by readers is not Transparent. 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 designed for human modification. Opaque formats include PostScript, PDF, 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 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.

A.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.

A.3 Copying in Quantity

If you publish printed copies 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 publicly accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. 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.

A.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:

- 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.

- 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 less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- 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.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled "History," and 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.
- 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.
- In any section entitled "Acknowledgements" or "Dedications," preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- 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.
- Delete any section entitled "Endorsements." Such a section may not be included in the Modified Version.
- Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

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.

A.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.

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."

A.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.

A.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, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate," and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they 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 quarter of the entire aggregate, the Document's Cover Texts

may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.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 provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense, or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.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.

A.11 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 © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License."

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

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.