
C# Coding Guidelines

Giuseppe Greco <giuseppe.greco@agamura.com>

2.0

Copyright © 2008 Agamura, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 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".

Table of Contents

Introduction.....	1
Source Files.....	2
Classes and Structs.....	2
Methods.....	3
Properties.....	3
Variables and Constants.....	4
Control Flow.....	5
The <code>if</code> Statement.....	5
The <code>for</code> Statement.....	5
The <code>foreach</code> Statement.....	5
The <code>while</code> Statement.....	5
The <code>do</code> Statement.....	6
The <code>switch</code> Statement.....	6
The <code>try-catch</code> Statement.....	7
Lexical Issues.....	7
Naming Convention.....	7
Indentation and White Space.....	8
Braces.....	9
Unstable Layout.....	9
A. Code Example.....	10
ThermometerConsole.cs.....	10
DetectTemperatureEventArgs.cs.....	11
Probe.cs.....	12
Thermometer.cs.....	14
B. GNU Free Documentation License.....	15

Introduction

This document describes a set of coding style guides for developing programs in the C# language.

A style guide is a set of mandatory requirements for layout and formatting. Uniform style makes it easier for other developers to grasp the essence of your programs quickly.

A style guide makes you more productive because it *reduces gratuitous choice*. If you do not have to make choices about trivial matters, you can spend your energy on the solution of real problems.

In these guidelines, several constructors are plainly outlawed. That does not mean that developers using them are evil or incompetent. It does mean that the constructs are not essential and can be expressed just as well or even better with other language constructors.

If you already have programming experience, you may be initially uncomfortable at giving up some fond habits. However, it is a sign of professionalism to set aside personal preferences in minor matters and to compromise for the benefit of the group.

Source Files

Source files should be kept as short as possible. Put every class in a separate file, and name that file like the class followed by the `.CS` extension.

For each assembly, create a new directory named like the assembly itself, and inside each assembly directory, map namespaces one-to-one to the directory layout. For example, the namespace *Thermota.Core.Properties* in the assembly *Thermota.Core* should be mapped to the directory `Thermota.Core/Properties`.

Organize the material in each source file as follows:

- File header
- `using` statements
- `namespace` statement, if appropriate
- `class`, `struct`, or `enum` element

The file header should at least include the file name, begin date, author, and copyright notice:

```
/** -----  
/** <sourcefile name="Thermometer.cs" language="C#" begin="17-Oct-2007">  
/**   <author name="Giuseppe Greco" email="giuseppe.greco@agamura.com"/>  
/**   <copyright company="Agamura" url="http://www.agamura.com">  
/**     Copyright (C) 2007 Agamura, Inc. All rights reserved.  
/**   </copyright>  
/** </sourcefile>  
/** -----
```

Classes and Structs

Each class and each struct should be preceded by a documentation comment explaining the purpose of the class or the purpose of the struct, respectively:

```
/// <summary>  
/// Models a thermometer.  
/// </summary>  
public class Thermometer  
{  
    ...  
}
```

Use the documentation feature supported by the C# compiler by including XML tags in special comment lines preceded by three slashes just before the code block they refer to.

Define features in alphabetical order and group them as following:

- `const` fields
- `readonly` fields
- `static` variables
- Instance (or class) variables
- Properties
- Operators
- Methods
- Nested classes

`public` features should precede `protected` and `private` features.

Leave a blank line after each group of related fields or variables as well as after every property or method.

All instance variables must be private. Use properties to provide access to them. (However, instance variables of a private inner class or struct may be public.) Methods, properties, and `const` fields can be either `public`, `protected`, or `private`, as appropriate.

All features must be tagged either `public`, `protected`, or `private`. Do not rely on the default visibility.

Methods

Every method should be preceded by a documentation comment explaining the purpose of the method:

```
/// <summary>
/// Handles the <see cref="Probe.DetectTemperature"/> event.
/// </summary>
/// <param name="sender">
/// The <see cref="Probe"/> that generated the event.
/// </param>
/// <param name="args">
/// A <see cref="DetectTemperatureEventArgs"/> that contains the last detected
/// temperature.
/// </param>
private void OnDetectTemperature(Object sender, DetectTemperatureEventArgs args)
{
    ...
}
```

Parameter names should be explicit, especially if they are integers or Booleans:

```
public void DetectTemperatureEventArgs(int i)
{
    // Avoid
}

public void DetectTemperatureEventArgs(int temperature)
{
    // OK
}
```

Of course, for very generic methods, short parameter names may be more appropriate:

```
public static void Sort(int[] i)
{
    // OK
}
```

Methods should have at most 30 lines of code. The method header, comments, blank lines, and lines containing only braces are not included in this count. Methods that consist of one long `if-else` or `switch` may be longer, provided each branch in 10 lines or fewer. This rule forces you to break up complex computations into separate methods.

Properties

Every property should be preceded by a documentation comment explaining the purpose of the property and describing the value it represents:

```
/// <summary>
/// Gets the detected temperature.
/// </summary>
```

```
/// <value>
/// An <see cref="Int32"/> that represents detected temperature in degrees
/// Celsius.
/// </value>
public int Temperature
{
    get { return temperature; }
    set { temperature = value; }
}
```

Variables and Constants

Do not define all variables at the beginning of a block:

```
public static void Main()
{
    string command; // Avoid
    bool exit = false; // Avoid
    Thermometer thermometer = new Thermometer();

    Console.WriteLine(MessageWelcome);

    do {
        command = Console.ReadLine();

        ...
    } while (!exit);
}
```

Instead, define each variable just before it is used the first time:

```
public static void Main()
{
    Thermometer thermometer = new Thermometer();

    Console.WriteLine(MessageWelcome);

    string command; // OK
    bool exit = false; // OK

    do {
        command = Console.ReadLine();

        ...
    } while (!exit);
}
```

Do not define two variables on the same line:

```
int celsius = 0, fahrenheit = 0; // Avoid
```

Define `const` fields as `private` or `protected` if no other classes have an interest in them.

Do not use *magic numbers*! A magic number is a numeric constant embedded in code, without a constant definition. Any number except -1, 0, and 1 is considered magic:

```
temperature = random.Next(-273, 100); // Avoid
```

Use `const` fields instead:

```
private const int AbsoluteZero = -273;
private const int MaxTemperature = 100;

...

temperature = random.Next(AbsoluteZero, MaxTemperature); // OK
```

Control Flow

The `if` Statement

Use the `if` statement to select a statement for execution based on the value of a Boolean expression:

```
if (temperatureSum > 0) {
    averageTemperature = temperatureSum / temperatures.Count;
}
else {
    averageTemperature = this.Temperature;
}
```

To keep you out of trouble, execution statements should always be enclosed within curly braces.

The `for` Statement

Use the `for` statement only when a variable runs from somewhere with some constant increment or decrement:

```
for (int i = 0; i < temperatures.Count; i++) {
    temperatureSum += (int) temperatures[i];
}
```

As for the `if` statement, execution statements should always be enclosed within curly braces.

The `foreach` Statement

Use the `foreach` statement to iterate through a collection that do not need to be changed:

```
foreach (int temperature in temperatures) {
    temperatureSum += temperature;
}
```

As for the `for` statement, execution statements should always be enclosed within curly braces.

Note

Using the `foreach` statement to change a collection may cause unpredictable side effects.

The `while` Statement

Use the `while` statement to execute a statement until a specified expression evaluates to `false`:

```
while (true) {
    if (DetectTemperature != null) {
        DetectTemperature(this, args);
    }
    ...
}
```

As for the `foreach` statement, execution statements should always be enclosed within curly braces.

Note

If the specified expression never evaluates to `true`, the statement will never be executed.

The `do` Statement

Use the `do` statement to execute a statement repeatedly until a specified expression evaluates to `false`, at least one time even if the expression never evaluates to `true`:

```
do {
    command = Console.ReadLine();

    switch (command) {
        ...

        case CommandExit:
            exit = true;
            break;
    }
} while (!exit);
```

As for the `while` statement, execution statements should always be enclosed within curly braces.

The `switch` Statement

Use the `switch` statement to handle multiple selections by passing control to one of the `case` statements within its body:

```
switch (command) {
    case CommandAverage:
        Console.WriteLine(thermometer.AverageTemperature);
        break;
    case CommandCurrent:
        Console.WriteLine(thermometer.Temperature);
        break;
    case CommandExit:
        exit = true;
        break;
    default:
        Console.WriteLine(CommandAverage + "\t\t" + MessageAverage);
        Console.WriteLine(CommandCurrent + "\t\t" + MessageCurrent);
        Console.WriteLine(CommandExit + "\t\t" + MessageExit);
        Console.WriteLine(CommandHelp + "\t\t" + MessageHelp);
        break;
}
```

The `case` statements must evaluate constant expressions, and the `switch` body should always end with a `default` statement handling unexpected options.

The try-catch Statement

Use the `try-catch` statement to handle raised exceptions:

```
try {
    temperature = random.Next(AbsoluteZero, MaxTemperature);
}
catch (ArgumentOutOfRangeException) {
    temperature = MaxTemperature;
}
```

If necessary, use `finally` to clean up any resources allocated in the `try-catch` block:

```
try {
    temperatures.Add(temperature);
}
catch (ArgumentOutOfRangeException e) {
    ...
}
finally {
    probe.Stop();
}
```

Note

Control is always passed to the `finally` block regardless of how the `try-catch` block exits.

Lexical Issues

Naming Convention

Before you go through the rules for naming program elements, you should be aware of the two most common ways of using character casing (UPPER and lower case) to distinguish between elements. They are:

- | | |
|----------------------|--|
| Pascal Casing | The first character is upper case, and the first letter of each word is also upper case. All other characters are lower case; for example, <code>CurrentTemperature</code> . |
| Camel Casing | The first character is not upper case, but the first letter of each word is upper case. All other characters are lower case; for example, <code>currentTemperature</code> . |

The following rules specify how to define identifier names:

- **Namespaces** must be named using Pascal casing, and their names must be nouns or noun phrases describing a logical group of classes (do not use any prefix); for example, `Thermometer`.
- **Classes** and **structs** must be named using Pascal casing, and their names must be nouns or noun phrases describing behavior (do not use any prefix); for example, `Thermometer`. **Exception classes** must always have the `Exception` suffix; for example, `OutOfRangeException`.
- **Interfaces** must be named using Pascal casing, and their names must be nouns or noun phrases prefixed with `I` describing behavior; for example, `IComponent`.
- **Enumeration** types and values must be named using Pascal casing, and their names must be singular nouns or plural nouns for bit fields (do not use any prefix); for example, `TemperatureScale.Celsius`.
- **ReadOnly** and **constant** fields must be named using Pascal casing, and their names must be nouns or abbreviations for nouns; for example, `AbsoluteZero`.
- **Parameter** and **non-constant** fields must be named using Camel casing, and their names should be nouns or noun phrases describing the meaning of the field; for example, `newTemperature`.

- **Variables** must be named using Camel casing, and their names should describe the meaning of the variable, unless they are used in trivial counting loops; for example, `sumTemperature`.
- **Methods** must be named using Pascal casing, and their names must be verbs or verb phrases; for example, `OnDetectTemperature()`.
- **Properties** must be named using Pascal casing, and their names must be nouns or noun phrases describing the meaning of the property (consider naming properties like the attributes they represent); for example, `AverageTemperature`.
- **Events** must be named using Pascal casing, and their name should be verbs or verb phrases describing the event; for example, `DetectTemperature`.
- **Event handlers** must be named using Pascal casing, and their names should be the same as the name of the events they handle with the `EventHandler` suffix. Event handlers must always have two parameters named `sender` and `args`, respectively, and event argument classes should always have the `EventArgs` suffix:

```
public delegate void DetectTemperatureEventHandler(
    Object sender, DetectTemperatureEventArgs args);
```

Summary.

	Pascal Casing	Camel Casing	Comments
Namespaces	X		
Classes/Structs	X		
Exception Classes	X		End with Exception
Event Argument Classes	X		End with EventArgs
Interfaces	X		Start with I
Enumerations	X		
Enumeration Values	X		
ReadOnly/Const Fields	X		
Public Fields	X		
Protected Fields		X	
Private Fields		X	
Parameters		X	
Methods	X		
Properties	X		
Events	X		
Event Handlers	X		End with EventHandler

Indentation and White Space

Use tab stops every four columns and set them to be expanded to spaces. That means you will need to change the tab stop setting in your editor!

Use blank lines freely to separate parts of a method that are logically distinct. Use a blank space around every binary operator:

```
averageTemperature=temperatureSum/temperatures.Count; // Avoid
averageTemperature = temperatureSum / temperatures.Count; // OK
```

Leave a blank space after (and not before) each comma, semicolon, and keyword, but not after a method name:

```
public void OnDetectTemperature(Object sender, DetectTemperatureEventArgs args)
{
    if (temperatures.Count == MaxCapacity) {
        temperatures.RemoveAt(0);
    }
    ...
}
```

Whenever possible, every line should fit on 80 columns. If you must break a statement, add an indentation level for the continuation:

```
probe.DetectTemperature +=
    new EventHandler<DetectTemperatureEventArgs>(OnDetectTemperature);
```

When breaking arithmetic expressions, start the indented line with an operator:

```
fahrenheit = (celsius * 1.8)
             + 32;

celsius = (fahrenheit - 32)
          * (5 / 9);
```

Braces

Opening curly braces should be placed on the same line as the keyword with which they are associated with a preceding space, unless you are declaring a namespace, class, struct, interface, property, or method. In these cases opening curly braces should begin on a new line. Keywords and closing curly braces must always line up vertically:

```
namespace Thermota.Core
{
    public class Probe
    {
        private void Detect()
        {
            while (true) {
                if (DetectTemperature != null) {
                    DetectTemperature(this, args);
                }
                ...
            }
        }
    }
}
```

Unstable Layout

Some programmers take great pride in lining up certain columns in their code:

```
private const int AbsoluteZero = -273;
private const int MaxTemperature = 100;
```

This is undeniably neat, but the layout is not *stable* under change. A new constant name that is longer than the preallotted number of columns requires that you move *all* entries around:

```
private const int AbsoluteZero      = -273;
private const int MaxTemperature    = 100;
private const int DetectionInterval = 5000;
```

This is just the kind of trap that makes you decide to use a name like `DetectionInt` instead, which is less clear.

A. Code Example

This appendix presents the full source code from which the examples in the previous sections have been extracted. The complete example project can be downloaded from Agamura's developer site [<http://developer.agamura.com/resources/>].

ThermometerConsole.cs

```
#region file description
/* -----
/* <sourcefile name="ThermometerConsole.cs" language="C#" begin="17-Oct-2007">
/*   <author name="Giuseppe Greco" email="giuseppe.greco@agamura.com"/>
/*   <copyright company="Agamura" url="http://www.agamura.com">
/*     Copyright (C) 2007 Agamura, Inc. All rights reserved.
/*   </copyright>
/* </sourcefile>
/* -----
#endregion

#region using statemets
using System;
using System.Threading;
using Thermota.Core;
using Thermota.Properties;
#endregion

namespace Thermota
{
    /// <summary>
    /// Implements the thermometer console.
    /// </summary>
    public class ThermometerConsole
    {
        #region private fields
        private const string Prompt = "[thermota ~]$";

        private static readonly string CommandAverage;
        private static readonly string CommandCurrent;
        private static readonly string CommandExit;
        private static readonly string CommandHelp;

        private static readonly string MessageWelcome;
        private static readonly string MessageAverage;
        private static readonly string MessageCurrent;
        private static readonly string MessageExit;
        private static readonly string MessageHelp;
        #endregion

        #region private constructors
        /// <summary>
        /// Initializes static members.
        /// </summary>
        static ThermometerConsole()
        {

```

```
// Commands
CommandAverage = Resources.Command_Average;
CommandCurrent = Resources.Command_Current;
CommandExit = Resources.Command_Exit;
CommandHelp = Resources.Command_Help;

// Messages
MessageWelcome = Resources.Message_Welcome;
MessageAverage = Resources.Message_Average;
MessageCurrent = Resources.Message_Current;
MessageExit = Resources.Message_Exit;
MessageHelp = Resources.Message_Help;
}
#endregion

#region public methods
/// <summary>
/// Entry point of the program.
/// </summary>
public static void Main()
{
    Thermometer thermometer = new Thermometer();

    Console.WriteLine(MessageWelcome);

    string command;
    bool exit = false;

    do {
        Console.Write("\n" + Prompt + " ");
        command = Console.ReadLine();

        if (command == CommandAverage) {
            Console.WriteLine(thermometer.AverageTemperature);
        }
        else if (command == CommandCurrent) {
            Console.WriteLine(thermometer.Temperature);
        }
        else if (command == CommandExit) {
            exit = true;
        }
        else {
            Console.WriteLine(CommandAverage + "\t\t" + MessageAverage);
            Console.WriteLine(CommandCurrent + "\t\t" + MessageCurrent);
            Console.WriteLine(CommandExit + "\t\t" + MessageExit);
            Console.WriteLine(CommandHelp + "\t\t" + MessageHelp);
        }
    } while (!exit);
}
#endregion
}
```

DetectTemperatureEventArgs.cs

```
#region file description
/** -----
/** <sourcefile name="DetectTemperatureEventArgs.cs" language="C#" begin="17-Oct-2007">
/** <author name="Giuseppe Greco" email="giuseppe.greco@agamura.com"/>
/** <copyright company="Agamura" url="http://www.agamura.com">
/** Copyright (C) 2007 Agamura, Inc. All rights reserved.
/** </copyright>
/** </sourcefile>
/** -----
#endregion

#region using statements
using System;
#endregion
```

```
namespace Thermota.Core
{
    /// <summary>
    /// Provides data for the <see cref="Probe.DetectTemperature"/> event.
    /// </summary>
    public class DetectTemperatureEventArgs : EventArgs
    {
        #region private fields
        private int temperature;
        #endregion

        #region public properties
        /// <summary>
        /// Gets the detected temperature.
        /// </summary>
        /// <value>
        /// An <see cref="Int32"/> that represents detected temperature in
        /// degrees Celsius.
        /// </value>
        public int Temperature
        {
            get { return temperature; }
        }
        #endregion

        #region public constructors
        /// <summary>
        /// Initializes a new instance of the <see cref="DetectTemperatureEventArgs"/>
        /// class with the specified temperature.
        /// </summary>
        /// <param name="temperature">
        /// An <see cref="Int32"/> that represents the detected temperature in
        /// degrees Celsius.
        /// </param>
        public DetectTemperatureEventArgs(int temperature)
        {
            this.temperature = temperature;
        }
        #endregion
    }
}
```

Probe.cs

```
#region file description
/** -----
/** <sourcefile name="Probe.cs" language="C#" begin="17-Oct-2007">
/**   <author name="Giuseppe Greco" email="giuseppe.greco@agamura.com"/>
/**   <copyright company="Agamura" url="http://www.agamura.com">
/**   Copyright (C) 2007 Agamura, Inc. All rights reserved.
/**   </copyright>
/** </sourcefile>
/** -----
#endregion

#region using statements
using System;
using System.Threading;
#endregion

namespace Thermota.Core
{
    /// <summary>
    /// Implements a virtual probe for simulating temperature detections.
    /// </summary>
    public class Probe
    {
        #region private fields
```

```
private const int AbsoluteZero = -273;
private const int MaxTemperature = 100;
private const int DetectionInterval = 5000;
private Thread detector;
#endregion

#region public events
/// <summary>
/// Occurs when a temperature detection is performed.
/// </summary>
public event EventHandler<DetectTemperatureEventArgs> DetectTemperature;
#endregion

#region public constructors
/// <summary>
/// Initializes a new instance of the <see cref="Probe"/> class.
/// </summary>
public Probe() { }
#endregion

#region public methods
/// <summary>
/// Starts temperature detections.
/// </summary>
public void Start()
{
    detector = new Thread(new ThreadStart(Detect));
    detector.Start();
    detector.IsBackground = true;
}

/// <summary>
/// Stops temperature detections.
/// </summary>
public void Stop()
{
    detector.Join(0);
}
#endregion

#region private methods
/// <summary>
/// Simulates temperature detections.
/// </summary>
private void Detect()
{
    int temperature;
    Random random = new Random();

    while (true) {
        try {
            temperature = random.Next(AbsoluteZero, MaxTemperature);
        }
        catch (ArgumentOutOfRangeException) {
            temperature = MaxTemperature;
        }

        DetectTemperatureEventArgs args = new DetectTemperatureEventArgs(temperature);

        if (DetectTemperature != null) {
            DetectTemperature(this, args);
        }

        Thread.Sleep(DetectionInterval);
    }
}
#endregion
}
}
```

Thermometer.cs

```
#region file description
/** -----
/** <sourcefile name="Thermometer.cs" language="C#" begin="17-Oct-2007">
/**   <author name="Giuseppe Greco" email="giuseppe.greco@agamura.com"/>
/**   <copyright company="Agamura" url="http://www.agamura.com">
/**     Copyright (C) 2007 Agamura, Inc. All rights reserved.
/**   </copyright>
/** </sourcefile>
/** -----
#endregion

#region using statements
using System;
using System.Collections;
#endregion

namespace Thermota.Core
{
    /// <summary>
    /// Models a thermometer.
    /// </summary>
    public class Thermometer
    {
        #region private fields
        private const int MaxCapacity = 50;

        private Probe probe;
        private int temperature;
        private ArrayList temperatures;
        #endregion

        #region public properties
        /// <summary>
        /// Gets the current temperature.
        /// </summary>
        /// <value>
        /// An <see cref="Int32"/> that represents the current temperature in
        /// degrees Celsius.
        /// </value>
        public int Temperature
        {
            get { return temperature; }
        }

        /// <summary>
        /// Gets the average temperature.
        /// </summary>
        /// <value>
        /// An <see cref="Int32"/> that represents the average temperature in
        /// degrees Celsius.
        /// </value>
        public int AverageTemperature
        {
            get
            {
                int temperatureSum = 0;

                foreach (int temperature in temperatures) {
                    temperatureSum += temperature;
                }

                int averageTemperature;

                if (temperatureSum > 0) {
                    averageTemperature = temperatureSum / temperatures.Count;
                }
                else {
                    averageTemperature = this.Temperature;
                }
            }
        }
    }
}
```

```
        return averageTemperature;
    }
}
#endregion

#region public constructors
/// <summary>
/// Initializes a new instance of the <see cref="Thermometer"/> class.
/// </summary>
public Thermometer()
{
    temperatures = ArrayList.Synchronized(new ArrayList(MaxCapacity));
    probe = new Probe();

    probe.DetectTemperature +=
        new EventHandler<DetectTemperatureEventArgs>(OnDetectTemperature);

    probe.Start();
}
#endregion

#region private methods
/// <summary>
/// Handles the <see cref="Probe.DetectTemperature"/> event.
/// </summary>
/// <param name="sender">
/// The <see cref="Probe"/> that generated the event.
/// </param>
/// <param name="args">
/// A <see cref="DetectTemperatureEventArgs"/> that contains the last
/// detected temperature.
/// </param>
private void OnDetectTemperature(Object sender, DetectTemperatureEventArgs args)
{
    temperature = args.Temperature;

    if (temperatures.Count == MaxCapacity) {
        temperatures.RemoveAt(0);
    }

    temperatures.Add(temperature);
}
#endregion
}
}
```

B. GNU Free Documentation License

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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. Secondly, 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.

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 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.

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.

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 1.2 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.