This document describes some basic steps to develop and test \LaTeX. It is meant for newcomers to the project or people who want to evaluate \LaTeX by inspecting the sources.
# Contents

1. **Introduction** 7  
   1.1. Audience ........................................ 7  
   1.2. Mailing Lists ..................................... 7  
   1.3. Organizational Agreements ..................... 8  
      1.3.1. Language .................................. 8  
      1.3.2. Maintainers of Files ................. 8  

2. **Prerequisites** 9  
   2.1. User Account at Berlios ....................... 9  
   2.2. Java ........................................... 10  
   2.3. TEXMF .......................................... 10  
   2.4. CVS Client .................................... 11  
   2.5. A Command Line Interpreter .................. 11  
   2.6. Ant ............................................ 11  
   2.7. Perl ............................................ 11  

3. **The Development Environment** 13  
   3.1. Eclipse .......................................... 13  
      3.1.1. Eclipse Installation .................... 13  
      3.1.2. Downloading the Sources ............ 15  
      3.1.3. Configuring Eclipse ................... 17  
      3.1.4. Code Templates ......................... 19  
      3.1.5. The Code Formatter ..................... 19  
      3.1.6. Checkstyle ................................ 20  
      3.1.7. Spelling ................................... 23  
      3.1.8. Compiling \( \LaTeX \) ................... 24  
      3.1.9. Running \( \LaTeX \) ....................... 24  
      3.1.10. Committing Changes ................... 25  
      3.1.11. Running Ant from within Eclipse .... 26  
      3.1.12. Creating Javadoc ....................... 26  
      3.1.13. Creating the Installer ............... 28  
   3.2. Command Line Use .............................. 28  
      3.2.1. Downloading the Sources ............ 28  
      3.2.2. Checkstyle ................................ 28  
      3.2.3. Ant .......................................... 28  
      3.2.4. Compiling \( \LaTeX \) ................... 30
# Contents

3.2.5. Running \( \varepsilon \text{TEX} \) .............................................. 30  
3.2.6. JUnit ................................................................. 31  
3.2.7. Creating Javadoc .................................................. 32  
3.2.8. Creating the Installer ............................................ 33  
3.3. Use with Emacs and JDEE ......................................... 33  
3.4. Modelling: Jude ...................................................... 34  

4. Source Code Documentation ........................................... 35  
4.1. Javadoc ................................................................. 35  
4.2. Documentation of Primitives ..................................... 36  

5. Quality Assurance and Unit Tests .................................. 37  
5.1. Deficiencies of the Trip Test ..................................... 37  
5.2. Anatomy of a JUnit Test Class ................................... 37  
5.3. Creating Test Cases for the Interpreter ......................... 39  

6. The Source Tree Organization ...................................... 41  
6.1. The Toplevel Directory ........................................... 41  
6.2. develop: Development Support .................................. 42  
6.3. doc: Documentation ................................................ 42  
6.4. lib: Third-Party Libraries ....................................... 42  
6.5. src: The Sources ..................................................... 42  
6.6. util: Utilities ........................................................ 42  
6.7. work: User’s Working Area ....................................... 42  
6.8. www: The Web Site .................................................. 43  

7. Design Details .......................................................... 45  
7.1. Writing New Primitives ............................................ 45  
7.1.1. Executable Code .................................................. 45  
7.1.2. Registering New Macros ....................................... 46  
7.1.3. Registering New Macros Dynamically ....................... 47  
7.1.4. Exceptions ........................................................ 47  
7.1.5. Assignments ....................................................... 47  
7.1.6. Expandable Code ................................................ 48  
7.1.7. Conditionals – Also Called Ifs ................................ 49  
7.1.8. Interaction With Other Macros ................................ 50  

8. The Web Pages .......................................................... 51  
8.1. Overview ............................................................. 51  
8.2. Layout ................................................................. 52  
8.3. Automatic Generation ............................................. 53  

9. Licenses for \( \varepsilon \text{TEX} \) ........................................ 55  
9.1. Acceptable Licenses ................................................ 55  
9.2. Non-Acceptable Licenses .......................................... 56
A. Licenses

A.1. GNU Free Documentation License

57
1. Introduction

\(\LaTeX\) aims at providing a high-quality typesetting system. The development of \(\LaTeX\) has been inspired by the experiences with \TeX. The focus lies on an open design and a high degree of configurability. Thus \(\LaTeX\) should be a good base for further development.

On the other hand we have to take care not to leave the current user base of \TeX behind. \pdfTeX has taught us that a migration path from \TeX has a positive value in it. In the mean time the majority of \TeX users applies in fact \pdfTeX.

To provide a backward compatibility of \(\LaTeX\) with \TeX one special configuration is provided. Thus backward compatibility is just a matter of configuration.

1.1. Audience

This document is meant for developers and those interested in the sources and development processes of \(\LaTeX\). It should contain all information for getting started quickly.

1.2. Mailing Lists

If you are ready to try \(\LaTeX\) you might as well want to join a mailing list to get in contact with the community. The following mailing lists might be of interest:

**extex@dante.de**
A general mailing list about \(\LaTeX\). It has low traffic and is mainly in German. Subscribe and unsubscribe via the Web form [http://www.dante.de/listman/extex](http://www.dante.de/listman/extex).

**extex-eng@dante.de**
A general mailing list about \(\LaTeX\). It has currently very low traffic and is in English. Subscribe and unsubscribe via the Web form [http://www.dante.de/listman/extex-eng](http://www.dante.de/listman/extex-eng).

**extex-devel@dante.de**
A mailing list for the exchange of the developers of \(\LaTeX\). It has low traffic and is partly in German. Subscribe and unsubscribe via the Web form [http://www.dante.de/listman/extex-devel](http://www.dante.de/listman/extex-devel).

**extex-cvs@list.berlios.de**
A mailing list for automatic notification about changes in the CVS repository of


1. Introduction

\texttt{\LaTeX}. It is not meant to post mails on this list. This list is not archived. Subscribe and unsubscribe via the Web form \url{https://lists.berlios.de/mailman/listman/extex-cvs}. You need to be logged in at Berlios when registering.

\texttt{extex-bugs@list.berlios.de}

A mailing list for bug messages of \texttt{\LaTeX}. Subscribe and unsubscribe via the Web form \url{http://lists.berlios.de/mailman/listman/extex-bugs}.

1.3. Organizational Agreements

The developers of \texttt{\LaTeX} have agreed on some rule for cooperation. Those rules are documented here.

1.3.1. Language

The official project language for \texttt{\LaTeX} is English in the US dialect. The sources are documented in this language and the major documents are written in this language.

Since some of the developers are German this language might slip in during intensive discussions.

1.3.2. Maintainers of Files

Each file has a single maintainer – even if there are several authors. The maintainer has to be informed and has to agree on any changes in the file. The maintainership is usually indicated in the Java sources with the help of the tag \texttt{@author}. The first author is always the maintainer.

Changes to a file can be carried out by the maintainer or delegated to somebody else. The maintainer can change if both the old and new maintainer agree in this.
2. Prerequisites

2.1. User Account at Berlios

To commit changes to the repository you have to be enlisted as a developer for \( \varepsilon \LaTeX \). A first requirement for this is an account at Berlios – the hosting site. If you just want to read the sources then you can use anonymous access.

To register at Berlios use the page \url{http://developer.berlios.de/} and select the item on the upper left side. You will find yourself in the registration page as shown in figure 2.1. You will find your way through easily.

![New Account at Berlios](image)

Figure 2.1.: New Account at Berlios

When you have an account at Berlios you might be added to the developers list of \( \varepsilon \LaTeX \). This is to be done by one of the admins of \( \varepsilon \LaTeX \).
2. Prerequisites

2.2. Java

You need to have Java 1.4.2 or later installed on your system. You can get Java for several systems directly from java.sun.com. Download and install it according to the installation instructions for your environment.

To check that you have an appropriate Java on your path you can use the command `java` with the argument `-version`. This can be seen in the following listing:

```
# java -version
java version "1.4.2_09"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_09-b03)
Java HotSpot(TM) Client VM (build 1.4.2_09-b03, mixed mode)
```

Free Java implementations are currently not supported. They might work, but the last time we checked it, GCJ didn’t suffice. We would be happy to have someone working on a compatibility layer for a free Java implementation.

2.3. TEXMF

If you want to use more than the pure \texttt{\LaTeX} engine, fonts and macros can be inherited from a texmf tree. \texttt{\LaTeX} itself does not contain a full texmf tree. It comes just with some rudimentary files necessary for testing. Thus you should have installed a texmf tree, e.g. from a \texttt{\LaTeX}Live installation. This can be found on the Comprehensive \texttt{\LaTeX} Archive Network (CTAN).

There is no need to install the texmf tree in a special place. You have to tell \texttt{\LaTeX} anywhere where it can be found. It is even possible to work with several texmf trees.

One requirement for the texmf trees is that they have a file database (\texttt{1s-R}). \texttt{\LaTeX} can be configured to work without it, but then \texttt{\LaTeX} is deadly slow. Thus you do not really want to try this alternative.

To use your texmf tree you should create a configuration file in your home directory. On a Unix system the home directory is stored the environment variable \texttt{$HOME$}. On a Windows system the home directory is usually located under \texttt{C:\WINDOWS\Profiles\}. The configuration file must have the name \texttt{.extex} (a little intelligence test under Windows;-). It contains one line of the following form

```
texmf.path=/usr/lib/texmf
```

The value should point to the location of the texmf tree. If you have several texmf trees which need to be used you can put them into this attribute by separating them by a platform-specific separator. This separator is a colon (:) under Unix and a semicolon (;) under Windows.
2.4. CVS Client

You need a CVS client installed. In the simplest case this is the client build into the IDE Eclipse, or the command line version of cvs.

2.5. A Command Line Interpreter

For several tasks it is convenient to have a command line interpreter at hand. On Unix this can be the (bourne, bash,...) shell. On Windows we recommend the Cygwin suite which contains the bash.

2.6. Ant

Ant is used throughout the whole build system. For most purposes it is sufficient to live with the files distributed. Nevertheless it might be necessary to have ant installed and on the path for special tasks.

2.7. Perl

Perl is used to create the wae site. Thus it has to be installed for this purpose. For the usual development it is not necessary to have Perl installed.
2. Prerequisites
3. The Development Environment

There is no mandatory IDE for the development of $\LaTeX$. Nevertheless in practice you can get good support if you stick to the development environment widely used within the $\LaTeX$ community. This is based on the Eclipse IDE.

3.1. Eclipse

Eclipse is a free IDE for Java and other programming languages. It also provides a framework for the development of own programs. But this is not needed for the $\LaTeX$ core. Currently the version 3.1 of Eclipse is used within the $\LaTeX$ development team.

![Eclipse](image)

Figure 3.1.: Eclipse

3.1.1. Eclipse Installation

Eclipse can be downloaded for free from [http://www.eclipse.org](http://www.eclipse.org). There you can get a file appropriate for your operating system containing the software development kit (SDK). For instance

**eclipse-SDK-3.1-win32.zip**

for any decent Windows platform.

**eclipse-SDK-3.1-linux-gtk.zip**

for Linux on Intel x86 with GTK.
3. The Development Environment

Download the appropriate file and unpack it in the installation directory. A new subdirectory `eclipse` will be created containing all files of Eclipse. You are done with the basic installation. You can start the `eclipse` executable found in the just installed directory.

![Figure 3.2: Eclipse Workspace](image)

When Eclipse starts you first see the splash screen shown in figure 3.1. Then Eclipse requests a workspace – as shown in figure 3.2. The workspace is a directory where the projects live and where your preferences are stored. If you have chosen the workspace directory carefully, you can turn on the check mark in this dialog to be not asked again.

Finally you end up in the welcome window of Eclipse shown in figure 3.3. Take some time and read the introductory material found there.

![Figure 3.3: Eclipse Initial Window](image)
3.1. Eclipse

The following sections describe some of the configurations which should be performed in order to work with Eclipse on \(\varepsilon\TeX\).

3.1.2. Downloading the Sources

Now we are ready to create a project for the sources of \(\varepsilon\TeX\). Everything needed can be found in the CVS repository of \(\varepsilon\TeX\) hosted by Berlios. Thus we start to get things onto the local host. For this purpose we need to open a new perspective in Eclipse. A perspective is a collection of windows which are usually meant for a common task.

A new perspective can be opened via the window Window \(\rightarrow\) Open Perspective \(\rightarrow\) Other... which can be seen in figure 3.4(a). This menu item opens a dialog box which offers some perspectives for opening. Currently we need a “CVS Exploring” perspective. This perspective is meant for inspecting CVS repositories and manipulation. Thus this perspective is selected (see figure 3.4(b)) and the dialog is completed with the OK button.

![Open Perspective](a)

![Selecting “CVS Exploring Perspective”](b)

Figure 3.4: Switching to a Perspective

Now a CVS exploring perspective is opened (see figure 3.5). You see a lot of windows and icons there. The tab “CVS Repositories” on the left side shows all repository locations currently known. This list is empty since we have not added any CVS locations yet.

To add a new repository location press the left mouse button on this tab and select New \(\rightarrow\) Repository Location... (see figure 3.6(a)). This brings up the dialog shown in figure 3.6(b). Here you can enter the coordinates of the \(\varepsilon\TeX\) CVS repository.

Note that you have to enter your account at Berlios and its password into the appropriate fields. If you do not have an account you can use the account name anonymous without any password to get reading access to the sources.

For this step you need online access to the internet. When the form is submitted with the OK button, the accessibility of the repository location is checked. Upon success the
3. The Development Environment

Figure 3.5.: CVS Exploring Perspective

(a) New Repository Location

(b) The Coordinates of the \( \varepsilon \)\TeX CVS Repository

Figure 3.6.: Adding the \( \varepsilon \)\TeX CVS Repository
new repository location is added to the list of repository locations as can be seen in figure 3.7(a).

![Repository Listed](image)

(a) The \texttt{ExTeX} Repository Listed

![Selecting to check-out](image)

(b) Selecting to check-out of \texttt{ExTeX}

Figure 3.7.: Checking-out of \texttt{ExTeX}

The next step consists of the check-out of the sources into an Eclipse project. To accomplish this you have to open the repository location and the HEAD within. Right-click the item \texttt{ExTeX} in the list (see figure 3.7(a)) and select Check out in the appearing context menu (see figure 3.7(b)). This will instruct Eclipse to create a new project in the workspace and fill it with the files from the repository.

Eclipse shows a progress bar during the check-out (see figure reffig:eclipse-checkout). This operation may take some time – we have been really busy creating files. When the checkout is finished you will find the project \texttt{ExTeX} in Eclipse containing the files within. The appearance of the Package View with those files is shown in figure 3.8(b).

### 3.1.3. Configuring Eclipse

Eclipse can be configured in a wide range. In the following sections some configuration options are proposed for the seamless development of \texttt{ExTeX}. The configuration is performed via the preferences dialog. This dialog can be opened via Window → Preferences. …

This menu brings up a dialog with many tabs which can be used to adjust the behaviour of Eclipse in many ways. The first step described below consists of the adaption of the appearance of the text editors. In the tree view on the left side of the dialog select General → Editors → Text Editors as shown in figure 3.9(b).

Now you can adjust some values on the right side of the dialog. Set the tag width to 8. Check the item Show print margin. Adjust the print margin to 80. And finally change the print color to red. The settings are stored in the workspace by accepting the settings with the OK button.

The rational is that the tabs should be used in the traditional sense of eight characters wide. In fact this is just a fallback. Usually tabs should be avoided where possible. The
3. The Development Environment

(a) The Checkout Progress Bar

(b) The $\LaTeX$ Project in the Package View

Figure 3.8.: Checking out $\LaTeX$ from the Repository

(a) Eclipse Preferences

(b) Eclipse Print Margin

Figure 3.9.: Some simple Settings

18
print margin of 80 is a weak rule. Try to limit yourself to this width. Sometimes it is not reasonable. Thus the checkstyle rules allow some more characters before complaining.

The following sections describe some more of the configuration options. You should really consider to follow the instructions to make maximal use of the configurations provided with \texttt{\LaTeX}.

### 3.1.4. Code Templates

Code templates provide a convenient way of filling in a frame for the documentation whenever some code is generated by Eclipse. The \texttt{\LaTeX} repository contains in the file \texttt{develop/eclipse/codetemplates.xml} some definitions of code templates. To import those definitions use the preferences (see figure 3.9(a)). Here select the item \texttt{Java $\rightarrow$ Code Style $\rightarrow$ Code Templates}. The button \texttt{Import...} can leads to a file selector where the file \texttt{develop/eclipse/codetemplates.xml} should be entered.

![Eclipse Preferences](image)

Figure 3.10.: Eclipse Preferences

After the code templates have been loaded a minor adaption is required. The entry under the key \texttt{Comments $\rightarrow$ Types} contains hard-wired a name and email address of the author. Here the own name and email address should be entered (see figure fig:eclipse-template-author).

### 3.1.5. The Code Formatter

Eclipse comes has a code formatter which can be invoked easily. This code formatter can be configured for different needs. A configuration for \texttt{\LaTeX} is contained in the repository under \texttt{develop/eclipse/formatter.xml}. In Eclipse the preference page can be found under the key \texttt{Java $\rightarrow$ Code Style $\rightarrow$ Formatter}. Here you can use the button
3. The Development Environment

![Image of Code Template Author Name](image)

Figure 3.11.: Code Template Author Name

Import... and select the configuration file. Now the profile "gene" is loaded and can be selected. This is shown in figure 3.12.

The formatter for Ant files has distinct parameters which should be adapted. The Preference page can be found under the key Ant → Editor → Formatter. The values should be adjusted as shown in figure 3.13.

3.1.6. Checkstyle

Checkstyle is a tool for checking the adherence of Java source code to certain rules. The rules can be freely configured. The \texttt{\LaTeX} repository contains a set of rules for checkstyle.

Checkstyle comes in a command line version and as a plug-in for Eclipse. This plugin has to be installed first.

Install Checkstyle plugin

To install the checkstyle plugin over the update wizard, use Help → Software Update → Find and Install → Search for new features to install → Next → New Remote Site and input the name 'checkstyle' and the URL \url{http://eclipse-cs.sourceforge.net/update} (see figure 3.14).

Configure Checkstyle

To set the configuration use Window → Preferences → Checkstyle. Create a new configuration and set the values in figure 3.15.

The configuration is stored in develop/eclipse/extex_checkstyle.xml.
3.1. Eclipse

Figure 3.12.: Settings for the Code Formatter

Figure 3.13.: Settings for the Ant Formatter
3. The Development Environment

Figure 3.14.: Checkstyle URL

Figure 3.15.: Checkstyle configuration
Enable Checkstyle

To enable the checks set in Project → Properties → Checkstyle the configuration to ExTeX (see figure 3.16).

![Figure 3.16: enable Checkstyle](image)

3.1.7. Spelling

Since English is not the native language of each developer it is a good idea to enable the spell checking of the source code. This feature is provided by Eclipse. In figure ref:eg:eclipse-spelling you can seen the preference page where you can activate the spell checking and provide a dictionary.

A dictionary can be got from SCOWL ([http://wordlist.sourceforge.net/](http://wordlist.sourceforge.net/)). You might want to use the US dictionary of medium size. Since this contains enough words to fit but not too much obscure words which hide typos.

After the spell checking is activated potential typos are marked in the editor with yellow lines. Correction proposals can be requested with the quick fix shortcut Ctrl-1.
3. The Development Environment

3.1.8. Compiling \LaTeX

Any source file in Eclipse is compiled automatically when the file is saved. Thus it is usually not necessary to compile things manually. If you feel the need to recompile everything you can achieve this by selecting Project → Clean... while the item Project → Build Automatically is checked (see figure 3.18).

![Figure 3.18: Recompiling a Project](image)

Another recompilation can be triggered via the Ant task compile.

3.1.9. Running \LaTeX

\LaTeX can be run from within Eclipse. We will describe here the execution of the compiled sources from a workspace. The execution of an external program would be an alternative. But this is only of minor relevance for a developer.

To run \LaTeX on some input file you have to create a run profile. The profile is kept and can be used the next time again. To create a run profile select the toolbar...
item Run... in the Java perspective (see figure 3.19(a)). In the appearing dialog select Java Application and press the button New. Now you can fill in the tabs as seen in figure 3.19(b). Enter a name, the project and the main class. The main class to use is de.dante.extex.main.TeX.

![The Run Menu](image1.png) ![Creating a run configuration](image2.png)

Figure 3.19.: Checking out \TeX from the Repository

On the Arguments tab you can enter the arguments for the invocation of \TeX. These are the same arguments which can also be used on the command line. Usually here the input file is given (see figure 3.20).

The Run button submits the command. A Console view is opened which can be used to interact with the the program – like in the command line interpreter.

### 3.1.10. Committing Changes

Eclipse ships with a CVS plugin which hides the details of the underlying version control system. Thus things are quite simple for the newcomers. On the other hand they are different from the procedure on the command line or other tools which mimic the command line (like WinCVS or TortoiseCVS).

The metaphor used in Eclipse is the synchronisation of the workspace with the repository. In the course of this synchronisation changes in the workspace files are committed to the repository, changes from the repository are updated into the workspace, and conflicts can be resolved. The conflict resolution – also known as merging – is the demanding task. Thus it has to be performed by a human.

To start the synchronisation select in the Package Manager or Navigator view the topmost \TeX node and activate in the context menu (right mouse button) the entry
3. The Development Environment

3.1.11. Running Ant from within Eclipse

To use Ant from within Eclipse you have to open the Ant view. This can be accomplished via the menu Window → Show View → Ant (see figure 3.22(a)).

In this view use the leftmost tool to add an Ant file. In the file selector choose the file ExTeX/build.xml. The Ant file is added to the (previously empty) list. It can be open to show the Ant target available (see figure 3.22(b)).

A double click on a target starts it's execution. The output is shown in a Console view.

A description of the targets can be found in section 3.2.3.

3.1.12. Creating Javadoc

To create the Javadoc HTML description of the sources you can use the Ant target javadoc. See sections 3.1.11 and 3.2.3. The result can be found in the directory target/javadoc.
3.1. Eclipse

Figure 3.21.: Starting Synchronization

(a) Opening an Ant View  
(b) The Ant View for \texttt{\LaTeX}

Figure 3.22.: Ant in Eclipse
3. The Development Environment

3.1.13. Creating the Installer

To create the installer you can use the Ant target installer. See sections 3.1.11 and 3.2.3. The result can be found in the file target/ExTeX-setup.jar.

3.2. Command Line Use

3.2.1. Downloading the Sources

The sources of \TeX{} are stored in a RCS repository. To access this repository you need access to the internet and RCS installed in some way.

The coordinates of the repository are:

- Connection type: pserver
- User: anonymous
- Host: cvs.extex.berlios.de
- Location: /cvsroot/extex
- Module: ExTeX

These coordinates allow you anonymous access to the sources with reading permissions only.

You need to download the sources of \TeX{}. On the command line this can be done with the following commands:

```
# cvs -d:pserver:anonymous@cvs.extex.berlios.de:/cvsroot/extex login
# cvs -z3 -d:pserver:anonymous@cvs.extex.berlios.de:/cvsroot/extex co ExTeX
```

If you want to participate in the development and are enlisted at Berlios you should use your account and password instead of the anonymous account.

3.2.2. Checkstyle

Checkstyle is a source code checker. \TeX{} should show a homogeneous appearance of the sources. Thus certain rules should be followed. Some of the rules are checked by the following command:

```
# build checkstyle
```

The result can be found in the file target/checkstyle.txt.

3.2.3. Ant

Apache Ant (http://ant.apache.org) is a build system for Java. It can be considered state of the art for Java programs to come with Ant scripts. \TeX{} supports Ant by providing a build.xml file for various tasks.

28
3.2. Command Line Use

The files needed for running Ant are included in the \TeX repository. Thus no additional installation is required. Just some setting need to be performed before Ant can be used.

An environment variable \texttt{JAVA\_HOME} should be defined which points to the JDK. The following jars should be placed on the environment variable \texttt{CLASSPATH}:

- $\texttt{JAVA\_HOME/lib/tools.jar}$
- $\texttt{JAVA\_HOME/lib/classes.zip}$
- and all jars found in develop/lib

The Ant can be invoked like in

```
$JAVA_HOME/bin/java -Dant.home=$ANT_HOME org.apache.tools.ant.Main compile
```

```
Buildfile: build.xml

compile:

BUILD SUCCESSFUL
Total time: 1 second
#
```

These steps are performed by the shell script \texttt{build} in the \TeX directory. Thus you can achieve the same effect – without any preparations except setting \texttt{JAVA\_HOME} – with the following command:

```
# build compile
```

```
Buildfile: build.xml

compile:

BUILD SUCCESSFUL
Total time: 1 second
#
```

The Ant configuration can be found in the file \texttt{build.xml} in the \TeX main directory. This configuration contains at least the following targets:

- **all** This target builds nearly everything.
- **compile** This target compiles all Java files of the sources into the directory \texttt{target/classes}. Note, that the test classess are not compiled. See also section 3.2.4.
- **jar** This target arranges that the file \texttt{target/extex.jar} is created. It contains the compiled sources of \TeX.
- **javadoc** This target creates the Javadoc HTML files in the directory \texttt{target/javadoc}. See also section 3.2.7.
3. The Development Environment

**checkstyle** This target applies checkstyle and creates a report in `target/checkstyle.txt`.

**tests** This target applies all JUnit test cases. See also section 3.2.6.

**installer** This target creates the installer with the graphical user interface. The result is placed in the file `target/ExTeX-setup.jar`. See also section 3.2.8.

**clean** This target deletes some generated files.

### 3.2.4. Compiling \(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X

Compiling \(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X form the command line can be accomplished with the help of the build script. The build script is a wrapper around Ant. It can be invoked with the following command:

```
# build compile
Buildfile: build.xml

compile:

BUILD SUCCESSFUL
Total time: 1 second
#
```

The generated files are placed in the sub-directory `target/classes`. Thus if this directory and the jars in `lib` are on the class path then \(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X can be run immediately.

### 3.2.5. Running \(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X

\(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X can be run with the help of the \(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X script in the main directory or by a direct invocation of Java. The start script is provided for Unix under the name `extex` and for Windows under the name `extex.bat`.

```
# extex work/empty.tex
This is ExTeX, Version 0.0 (ExTeX mode)
(work/empty)
No pages of output.
Transcript written on ./empty.log.
#
```

For the usual purposes these scripts can be used as a plug-in replacement for \(\mathbf{T}\)\(\varepsilon\)X. See the User’s Guide for the command line options.

To run \(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X from the command line prepare the class path – i.e. the environment variable `CLASSPATH` – to contain all libraries found in the directory `lib`. In addition the directory `target/classes` have to be on the class path. Then you can invoke \(\varepsilon\)\(\mathbf{T}\)\(\varepsilon\)X like in the following example:
The command line arguments are the same as for \texttt{extex} mentioned above.

### 3.2.6. JUnit

JUnit is the state of the art concerning test automation for Java programs. Thus \TeX provides some test cases in the form of JUnit classes.

All test can be run from the command line with the build script:

```bash
# build tests
Buildfile: build.xml

compile:
jar:
  [jar] Building jar: /home/gene/src/ExTeX/target/lib/extex.jar

compile.tests:
  [copy] Copying 148 files to /home/gene/src/ExTeX/target/classes

jar.tests:
  [jar] Building jar: /home/gene/src/ExTeX/target/lib/testsuite.jar

tests:
  [mkdir] Created dir: /home/gene/src/ExTeX/tmp/tests
  [junit] Running de.dante.extex.CurrentgrouplevelTest
  [junit] Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 1,623 sec
...

BUILD SUCCESSFUL
Total time: 3 minutes 54 seconds
#
```

This invocation runs all JUnit test cases found in the directory \texttt{src/test}. The results can be found in the directory \texttt{target/tests} with one file per test class.

To run single cases or a selected set of test cases you can use the parameter \texttt{cases}. This parameter is added to the command line arguments with the prefix \texttt{-D}. The value follows after an equals sign. The value is a pattern to select the test case files to be used. The pattern \texttt{**} denotes an arbitrary deep directory hierarchy. \texttt{*} denotes an arbitrary sequence of characters. Note that the pattern should end in \texttt{Test.java}. 

31
3. The Development Environment

Details on testing and test cases can be found in section 5.

3.2.7. Creating Javadoc

Creating the Javadoc HTML pages can best be accomplished with the help of the build script. Here the target javadoc does everything necessary:

```
# build javadoc
Buildfile: build.xml

generate:

javadoc:
  [javadoc] Generating Javadoc
  [javadoc] Javadoc execution
  [javadoc] Loading source files for package de.dante.extex...
  [javadoc] Loading source files for package de.dante.extex.color...
  [javadoc] Loading source files for package de.dante.extex.color.model...
...

BUILD SUCCESSFUL
Total time: 2 minutes 20 seconds
#
```

As the result of this invocation the Javadoc HTML pages are stored in the subdirectory target/javadoc.
3.2.8. Creating the Installer

The installer can be build with the help of the build script. The invocation looks as follows:

```bash
# build installer
Buildfile: build.xml

compile:

jar:
  [jar] Building jar: /home/gene/src/ExTeX/target/lib/extend.jar

installer:
  [izpack] Adding resource: IzPack.uninstaller ...
  [izpack] Setting the installer informations ...
  [izpack] Setting the GUI preferences ...
...
  [izpack] Writing Packs ...
  [izpack] Writing Pack #0: Core
  [izpack] Writing Pack #1: Libraries
  [izpack] Writing Pack #2: User Settings
  [izpack] Writing Pack #3: Documentation
  [izpack] Writing Pack #4: Fonts
  [izpack] Writing Pack #5: Sources

BUILD SUCCESSFUL
Total time: 1 minute 30 seconds
```

After the work is complete the installer can be found in the file `ExTeX-setup.jar` in the directory `target`. The use of the installer is described in the User’s Manual.

Alternatively the installer can also be created with the Ant task `installer`. Using this method can be applied from the command line and from within Eclipse.

Note that the installer is automatically created once a day and provided in the snapshot directory of the $\epsilon_X\text{TeX}$ Web site.

3.3. Use with Emacs and JDEE

JDEE is the extension of Emacs for the development with Java. $\epsilon_X\text{TeX}$ contains some support files for use in this context.

To be completed.
3. The Development Environment

3.4. Modelling: Jude

Jude is a UML modeller written in Java. It is distributed in a community edition for free use. Currently the version 1.6.2 is available from http://www.esm.jp/jude-web/index.html.

![Jude Screenshot](image)

Figure 3.23.: Jude

Jude should be used for any situations where UML diagrams are needed. A screenshot of Jude can be seen in figure 3.23.

Models for \( \varepsilon \)TEX should be placed in the directory doc/models.
4. Source Code Documentation

The source code has to be documented. TeX shows us a good example of a proper documentation. Donald Knuth has invented the Web system to keep together the documentation and the source code. The source code and documentation are extracted from a common file. In the Java world the Javadoc system has been invented for a similar purpose.

4.1. Javadoc

The Javadoc conventions for comments make it possible to extract the relevant part of the documentation and generate several output formats from it. The primary output format is HTML.
4. Source Code Documentation

4.2. Documentation of Primitives

The documentation of the primitives is contained in the Javadoc comments of the implementing Java classes. A script is used to extract the information from the sources for the User’s Manual. To make this happen, the documentation meant for the manual has to be marked and formatted specially.

To be completed.
5. Quality Assurance and Unit Tests

Quality assurance and testing play an important rôle in software development. Automated regression tests help to guarantee that functionality is preserved across releases.

5.1. Deficiencies of the Trip Test

Donald Knuth has provided the trip test for $\TeX$. This test is not suitable for $\varepsilon\TeX$ for several reasons:

- The trip test compares the log file and the dvi output. $\varepsilon\TeX$ does not guarantee identical log files.
- And $\varepsilon\TeX$ may produce more than dvi.
- The trip test covers only part of the functionality of $\TeX$. It is interesting to test other features too.
- The trip test contains tests for failures as well. In those cases the behaviour of $\varepsilon\TeX$ might be different.

As a consequence $\varepsilon\TeX$ comes with its own set of test cases.

5.2. Anatomy of a JUnit Test Class

To be completed.

```java
package de.dante.extex.interpreter.primitives;

import de.dante.test.ExTeXLauncher;

/**
 * This is a test suite for the primitive <tt>\relax</tt>.
 * @author <a href="mailto:gene@gerd-neugebauer.de">Gerd Neugebauer</a>
 */
```
5. Quality Assurance and Unit Tests

```java
public class RelaxTest extends ExTeXLauncher {

    public static void main(final String[] args) {
        junit.textui.TestRunner.run(RelaxTest.class);
    }

    public RelaxTest(final String arg) {
        super(arg);
    }

    public void test1() throws Exception {
        runCode("\relax",
               "--- log message ---
               \protect\"\%");
    }
}
```
5.3. Creating Test Cases for the Interpreter

To be completed.
5. Quality Assurance and Unit Tests
6. The Source Tree Organization

In this section the description of the directory hierarchy is contained. This structure is oriented on the structuring proposed by Maven (http://maven.apache.org).

6.1. The Toplevel Directory

The toplevel directory of an \LaTeX{} project contains certain files and sub-directories.

- **develop**
  The sub-directory *develop* contains bits and pieces needed for development only.

- **doc**
  The sub-directory *doc* contains documentation – papers written by the \LaTeX{} Group and material collected from elsewhere.

- **lib**
  The sub-directory *lib* contains libraries which need to be present for the final executable to run.

- **src**
  The sub-directory *src* contains the sources.

- **target**
  The sub-directory *target* contains the generated files. This directory is not present in the CVS archive.

- **tmp**
  The sub-directory *tmp* may contain intermediary files. This directory is not present in the CVS archive.

- **util**
  The sub-directory *util* contains some utilities for development. They are not included into the installer.

- **work**
  The sub-directory *work* may contain working files of single developers. It is not shared via the repository.

- **www**
  The sub-directory *www* contains the sources for the Web pages of \LaTeX.
6. The Source Tree Organization

6.2. develop: Development Support

   eclipse
   lib

6.3. doc: Documentation

   DevelopersGuide
   Library
   Publications
   UsersGuide
   models
   notes

6.4. lib: Third-Party Libraries

   This directory contains libraries needed for \LaTeX{} to run.

6.5. src: The Sources

   font
   java
   javadoc
   text

6.6. util: Utilities

   This directory contains various utilities and scripts.

   Installer

6.7. work: User’s Working Area

   Any user may have some files in the project area. Those files should not be committed to the Repository. For this purpose the directory work is reserved.
6.8. **www: The Web Site**

*src*
6. The Source Tree Organization
7. Design Details

This chapter contains some explanations, tips & tricks. It might be helpful to read them when you are concerned with the related topics.

7.1. Writing New Primitives

The core primitives of $\LaTeX$ are written in Java and bound to control sequences or active characters. In this section we will explain how to write new primitives in Java.

7.1.1. Executable Code

Executable primitives are those primitives which can be invoked in a left-hand-side context of the expansion. This is the case whenever the next top-level macro is treated. You can consider for example the treatment of the macro `\def` as such a case:

```
\def\abc{123}
```

In this example `\def` is an executable primitive.

Executable code has to implement the interface `Code`. Doing this directly is not hard. Nevertheless the abstract base class `AbstractCode` is provided which contains default implementations for all methods already. Thus only the interesting methods have to be overwritten.

In the simplest case only a constructor with one String argument and the method `execute()` has to be defined. Such an empty frame can be seen in the following example:

```java
package my.package;

import de.dante.extex.interpreter.context.Context;
import de.dante.extex.interpreter.Flags;
import de.dante.extex.interpreter.TokenSource;
import de.dante.extex.interpreter.exception.InterpreterException;
import de.dante.extex.interpreter.type.AbstractCode;
import de.dante.extex.typesetter.Typesetter;

class MyPrimitive extends AbstractCode {
    
    public MyPrimitive(final String name) {
        super(name);
        // initialization code -- if required
    }
}
```
7. Design Details

```java
public boolean execute(final Flags prefix,
            final Context context,
            final TokenSource source,
            final Typesetter typesetter
        ) throws InterpreterException {
    // implement the execution behaviour here
    return true;
}
```

In the method `execute()` you have access to other components. This can be utilized to implement the desired functionality. The following parameters are provided:

**Flags prefix** This parameter gives access to prefix arguments like `\immediate` or `global`. For this purpose the class Flags provides appropriate getters. You can even modify the flags passed to the method. Usually you should invoke `prefix.clear()` somewhere in your implementation when the prefix is not needed any more. If this method is omitted then the prefix is passed on to the next execution. This can be desirable if you want to implement a prefix primitive yourself.

**Context context** The context provides reading and writing access to the data stored in the processor. This information is the memory. It is written to file when dumping a format. Refer to the documentation of the interface Context for details.

**TokenSource source** The source provides access to the token stream. It can be used to get the next tokens if required. For example when implementing a primitive like `\def` it is necessary to read the next tokens as arguments: the macro name, the parameter pattern, and the expansion text. The token source can also be used to push tokens to the input stream to be read back in later. This feature is used when implementing expandable primitives.

**Typesetter typesetter** The typesetter is the component which collects nodes and finally sends them to the document writer. With access to this component it is possible to produce some output to the paper.

The return value indicates how to deal with prefix flags. The usual behaviour is to return `true`. This indicates that the flags should be cleared afterwards. For those primitives which modify the prefix flags the return value `false` must be used.

### 7.1.2. Registering New Macros

The primary way to register new macros is in the configuration file used by \LaTeX. For example the default file is located in the package `config` and named `extex.xml`. There you can find lines like the following one:
7.1. Writing New Primitives

To add another primitive to \(\varepsilon\)\TeX you should make a copy of this configuration file under a different name and add a line like the one shown above:

```xml
<define name="def"
class="de.dante.extex.interpreter.primitives.macro.Def"/>
```

Now you can invoke \(\varepsilon\)\TeX on the command line with the parameter -configuration or add a line extex.config to your .extex file pointing \(\varepsilon\)\TeX to your new configuration file:

```
extex -configuration config/myExTeX.xml
```

This is enough. In the instance of \(\varepsilon\)\TeX with these settings the new macro \texttt{myPrim} is defined and points to your code for execution.

7.1.3. Registering New Macros Dynamically

One extension provided with \(\varepsilon\)\TeX contains a dynamic definition of new macros. Those macros are defined at runtime. The assignment of the Java code to the macro name can be controlled with the help of a primitive. Check out whether the macro \texttt{javadoc} is defined in one of the configuration files provided and consult the documentation.

7.1.4. Exceptions

The implementing Java code for new primitives can signal abnormal situations with the help of exceptions. The exceptions used should be derived from InterpreterException, RuntimeExceptions and Errors or derived classes should not be used.

\(\varepsilon\)\TeX provides means for externalizing strings. Thus it should be made easy to translate the messages to other languages. For this purpose the class Localizer is provided. See the documentation of this class for details.

7.1.5. Assignments

Assignments are a special kind of executable code. \(\varepsilon\)\TeX defines that the parameter \texttt{\globaldef} is evaluated and the macro \texttt{\afterassignment} has some effect. To ease the development of assignments the abstract base class AbstractAssignment is provided. This class defines the method execute() appropriately. The only task left is to overwrite the method assign() to perform the assignment.

```java
package my.package;

import de.dante.extex.interpreter.context.Context;
import de.dante.extex.interpreter.Flags;
```
7. Design Details

```java
import de.dante.extex.interpreter.TokenSource;
import de.dante.extex.interpreter.exception.InterpreterException;
import de.dante.extex.interpreter.type.AbstractAssignment;
import de.dante.extex.typesetter.Typesetter;

class MyAssign extends AbstractAssignment {
    
    public MyAssign(final String name) {
        super(name);
        // initialization code -- if required
    }

    public void assign(final Flags prefix,
                       final Context context,
                       final TokenSource source,
                       final Typesetter typesetter
               ) throws InterpreterException {
        // implement the assignment here
    }
}
```

The arguments of the method `assign()` are the same as the arguments of `execute()` described above. In contrast to the remarks made there it is not necessary to return something. The clearing of the flags is done in the abstract class automatically.

### 7.1.6. Expandable Code

Some macros are expandable. This means that they can be used on the right-hand-side of an invocation as well. This feature is expressed by the interface `ExpandableCode`. Since Java does not allow multiple inheritance no abstract base class is provided.

To implement an expandable primitive it is sufficient to declare the interface for the class and implement the method `expand()`. This is sketched in the following example:

```java
package my.package;

import de.dante.extex.interpreter.context.Context;
import de.dante.extex.interpreter.Flags;
import de.dante.extex.interpreter.TokenSource;
import de.dante.extex.interpreter.exception.InterpreterException;
import de.dante.extex.interpreter.type(AbstractCode;
import de.dante.extex.interpreter.type.ExpandableCode;
import de.dante.extex.typesetter.Typesetter;

class MyExpandable extends AbstractCode implements ExpandableCode {
    
    public MyExpandable(final String name) {
        super(name);
    }
```
7.1. Writing New Primitives

```java
// initialization code -- if required
}

public boolean execute(final Flags prefix,
      final Context context,
      final TokenSource source,
      final Typesetter typesetter
) throws InterpreterException {
    // implement the execution behaviour here
    return true;
}

public void evaluate(final Flags prefix,
      final Context context,
      final TokenSource source,
      final Typesetter typesetter
) throws InterpreterException {
    // implement the evaluation behaviour here
}
}
```

The parameters of `evaluate()` are identical to those of `execute()`. But note, that the expected behaviour of `evaluate()` is that it does not modify the context or the typesetter but exclusively modifies the token source. Usually it reads some tokens and plugs back its result to the token stream.

7.1.7. Conditionals — Also Called Ifs

Conditionals are special because they modify the flow of control. In the macro programming language of \LaTeX{} this may lead to a mode where tokens are absorbed at high speed. In this mode it is necessary to identify conditionals to honor matching pairs of start and end tokens.

All necessary actions are performed by the abstract base class `AbstractIf`. The only thing to do is to implement the method `conditional()` which computes whether the then or the else branch should be considered relevant. This is shown in the following example:

```java
package my.package;

import de.dante.extex.interpreter.primitives.conditionals.AbstractIf;
import de.dante.extex.interpreter.context.Context;
import de.dante.extex.interpreter.Flags;
import de.dante.extex.interpreter.TokenSource;
import de.dante.extex.interpreter.exception.InterpreterException;
import de.dante.extex.typesetter.Typesetter;
```
7. Design Details

```java
class MyIf extends AbstractIf {

    public MyIf(final String name) {
        super(name);
        // initialization code -- if required
    }

    public boolean conditional(final Flags prefix,
                                final Context context,
                                final TokenSource source,
                                final Typesetter typesetter
            ) throws InterpreterException {
        // implement the evaluation of the conditional here
        return result;
    }
}
```

The parameters are the same as the parameters for `execute()` described above.

Note that any conditional is expandable automatically. Thus it should not modify the context or the typesetter.

7.1.8. Interaction With Other Macros

Several primitives of \texttt{\LaTeX} are implemented generically. Let us consider for example the macro \texttt{\the}. This primitive simply gathers the next token and delegates the task of providing an appropriate definition for \texttt{\the} to the definition of this token.

The ability to be usable after \texttt{\the} is expressed with the help of the interface \texttt{Theable}. Thus it is enough for a primitive to implement this interface if it needs to be usable after \texttt{\the}.

The following list contains some macros of \texttt{\LaTeX} and the related interfaces:

- \texttt{\advance} : Advanceable
- \texttt{\box} : Boxable
- \texttt{\count} : CountConvertible
- \texttt{\dimen} : DimenConvertible
- \texttt{\divide} : Dividable
- \texttt{\font} : FontConvertible
- \texttt{\multiply} : Multiplyable
- \texttt{\show} : Showable
- \texttt{\showthe} : Theable
- \texttt{\the} : Theable
8. The Web Pages

8.1. Overview

\(\varepsilon\)\TeX\ has a domain of its own. This domain \texttt{www.extex.org} has been registered by DANTE e.V. In this location the official Web pages (see figure 8.1) are provided.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig81.png}
\caption{www.extex.org}
\end{figure}

The Web pages are build with a simple generator for a Web site written in Perl. It has been made for \(\varepsilon\)\TeX. The aim is the ease of maintainance for normal content of pages. They are stored as simple HTML files and augmented automatically upon publication.

The layout is separated form the content and stored in several files. This makes it very easy to adapt the appearance without touching the contents.

The sources are kept in the subdirectory \texttt{src}. The generated files are put into the subdirectory \texttt{www}. Both locations can be configured.
8. The Web Pages

To generate the Web site run the following command, where the current directory is the directory www:

```
# make
```

This command creates a complete directory hierarchy with all necessary sub-directories in ../target/www. An exception are the directories named CVS. Those directories are ignored.

The files starting with . or ending in ~ or in .bak are also ignored. The files not ending in .html are copied into the destination tree. The files ending in .html are processed as follows: Text is inserted before the </head> tag from the file .headEnd. Text is inserted after the <body> tag from the file .bodyStart. Text is inserted before the </body> tag from the file .bodyEnd.

The text to be inserted is sought in the current directory and in case of failure upwards in the super-directories until it is found. In the inserted files the following entities and tags are replaced:

- `&top;` this is the relative path to the top directory.
- `&year;` this is the current year when generating.
- `&month;` this is the current month when generating.
- `&day;` this is the current day when generating.

- `<tabs/>` this is replaced by the contents of the file .tabs.
- `<navigation/>` this is replaced by the contents of the file .navigation.
- `<info/>` this is replaced by the contents of the file .info.

Note, that even so it looks like XML processing, currently the processing is based on string manipulation. Thus tricks possible with XML might not work here.

8.2. Layout

The current layout has the scheme shown in figure 8.2.

The Header contains the right aligned Logo only. It is the same on all pages. The Tab Bar shows the topmost navigation items with the Tab metaphor. The Navigation
8.3. Automatic Generation

The web pages are generated automatically every night. This task is performed with the help of a cron job on shell.berlios.de under the account gene. In the course of this generation the current sources are checked out from the CVS repository.

Thus the normal user simply has to edit the pages in the area www/src and check them into the CVS repository. The rest happens automagically.
8. The Web Pages
9. Licenses for $\varepsilon\chi$TeX

The project goal of $\varepsilon\chi$TeX is to provide a typesetting library facility. The global licence of the heart of $\varepsilon\chi$TeX is therefore the LGPL. But in order to keep open linking facility against all the different licenses we must be choose carefully third party program licenses, i.e. jar files.

9.1. Acceptable Licenses

The licence that we accept and why:

- LGPL
  for obvious reasons.

- modified BSD licence (http://www.xfree86.org/3.3.6/COPYRIGHT2.html#5)
  We can always convert this licence to LGPL.

- Apache 2.0 licence
  Unfortunately linking with GPL2 programs is in gray area. Interpretation differs between FSF and Apache but GPLv3 will be compatible.

- Public domain
  (obvious)

- GPL with explicit linking clause that don’t enforce viral clause of GPL and insure clear encapsulation of GPL license. It is the GPL with this special exception like this:

  As a special exception, if other files instantiate templates or use macros or inline functions from this file, or you compile this file and link it with other works to produce a work based on this file, this file does not by itself cause the resulting work to be covered by the GNU General Public License. However the source code for this file must still be made available in accordance with section (3) of the GNU General Public License.

  or like this:

  0
  Author: Bastien Roucaries
9. Licenses for \LaTeX

Linking this library statically or dynamically with other modules is making a combined work based on this library. Thus, the terms and conditions of the GNU General Public License cover the whole combination. As a special exception, the copyright holders of this library give you permission to link this library with independent modules to produce an executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module. An independent module is a module which is not derived from or based on this library. If you modify this library, you may extend this exception to your version of the library, but you are not obligated to do so. If you do not wish to do so, delete this exception statement from your version.

9.2. Non-Acceptable Licenses

The licence that don’t accept and why:

- **GPL license:**
  We would loose the free linking facility.

- **Apache 1.1 license:**
  It forbids linking with GPL software. Check if an upgrade to Apache 2.0 exists.

- **Eclipse public license:**
  It forbids linking with GPL software and has nasty side effects for proprietary software. Perhaps side effects exist with Apache 2.0 due to patents issue.

- **Original BSD license and Apache 1.0**
  Have a boring clause that forbids linking with a lot of different licenses – particularly GPL – and non carefully written proprietary software.
A. Licenses

A.1. GNU Free Documentation License

Version 1.2, November 2002
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.

Preamble

The purpose of this License is to strike a balance between maintaining the freedom to distribute free software and to copy and convey the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic,ältering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent abuse of author's intent.

This license restricts use of a particular distribution to the extent required to ensure a reasonabler expectation that the program modified by others will too be free. The precise meaning of this term is intended to include self-modification as well as modification by others.

The GNU General Public License is intended to make scientific research on free software truly feasible by ensuring hermetic, áltering the source code of such programs, and the need to prevent ab
A. Licenses

of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of the Document, to ensure that a person receiving it will be able to redistribute it under these terms. A secondary recipient is free to distribute the Document without requiring this. You may add a passage of up to five words from a Front-Cover Text as a Front-Cover Text of the Document, and include an item stating this fact in the Document's license notice.

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, and make the Modified Version available on an open network for public copying in the same form as the original publisher has made it available.

A. Use in the Title Page (and on the cover, 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 gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the adaptations 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, if it differs from that of the Document.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modification of the Document, stating this fact.

F. Indicate, immediately after the copyright notice, a notice giving the public permission to use the Modified Version under the terms of this License, including an indication of who has made the changes.

G. Preserve in the Modified Version the full list 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 its content stating at least the year, name, 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 year, name, and publisher of the Document as given on its Title Page, then add its content 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 Preserve its cover text. These may be placed in the "History" section. You may omit a network location for a work that was published within 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 and preserve its content in all copies. Add an appropriate copyright notice for your adaptation of that section, providing this License is included in the adaptation under the terms of section 4.

L. Preserve all the Invariant Sections of the Document, except for sections Entitled "Acknowledgements", "Endorsements", and "History".

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

N. Do not make a new version that adds sections Entitled "Endorsements" or "History".

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front matter sections or prefaces 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. This will affect which of the license notices in the Modified Version you must include.

If you add a section Entitled "Endorsements" as a Front-Cover Text of the Document, you must add an appropriate copyright notice to state that you do not add the text in question. You may also add a section Entitled "Endorsements" as a Back-Cover Text of the Document.

If you add a section Entitled "Endorsements" as a Back-Cover Text of the Document, you must also add an appropriate copyright notice to state that you do not add the text in question.

If you add a section Entitled "Endorsements" as a Back-Cover Text of the Document, you must also add an appropriate copyright notice to state that you do not add the text in question.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and copy the collection without changing the rights under this License. These combined works must be entirely free; thus, this License applies to the combination in the same way as to the original Document. When you produce an aggregate, 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".

7. AGGREGATION WITH INDEPENDENT WORKS

A preparation that includes an aggregate shall include the constituent works, except for those subsequently added by the aggregator and included in the aggregate.

A work that is included in an aggregate but is not itself an aggregate shall be treated as a constituent work in the 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 the License, as well as translations of parts of the text of the License for public benefit, but without permission.

In case of a disagreement between the translation of the original version of this License and the original version, the original version will prevail.

A section in the Document Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 3) will typically require changing the actual title.

9. TERMINATION

The text of this License contains copyrighted material. If you wish to copy, modify, and distribute this work, you must ensure that you satisfy the requirements set out in each section to the text of this License. You may not offer or distribute this text on a CD-ROM or similar physical medium without prior written permission from the copyright holder.
A.1. GNU Free Documentation License

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.

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 1.2 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:

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 small 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.