SGML for Windows NT

Setting up a free SGML/XML editing and publishing system on the Windows platform

Markus Hoenicka

hoenicka_markus@compuserve.com

SGML for Windows NT: Setting up a free SGML/XML editing and publishing system on the Windows platform

by Markus Hoenicka

Revision History

Revision 3.02 02-02-04
Various bugfixes
Revision 3.01 02-01-18
Updates for Emacs, PSGML, AucTeX
Revision 3.00 01-10-4
XML
Revision 2.13 01-3-7
Updates for PSGML, AucTeX, JadeTeX, OpenJade
Revision 2.12 00-10-24
GNU Free Documentation License
Revision 2.11 00-10-20
New Emacs directory layout, Cygwin, new JadeTeX instructions
Revision 2.10 00-4-28
Fixed links for Ghostscript and Ghostview
Revision 2.09 00-3-7
Update for Emacs 20.6, OpenJade 1.3, Ghostscript 6.01, Ghostview 2.9, fixed some broken links
Revision 2.08 00-2-10
Update for PSGML 1.2.1, Emacs 20.5.1, AucTeX 9.10p
Revision 2.07 99-9-7
Fixed AucTeX-related problems
Revision 2.06 99-8-22
Update for Emacs 20.4.1 and MiKTeX 1.20d
Revision 2.05 99-7-27
fixed the glitch in selecting the stylesheet with the DSSSL-File Options dialog
Revision 2.04 99-7-26
fixed the psgml-jade installation bug; extended the dvips config; fixed dependency on the config.ps in PSGML-Jade
Revision 2.03 99-7-21
just another bugfix, at increasing frequency?
Revision 2.02 99-7-18
another weekly(?) bugfix release
Revision 2.01 99-7-10
weeded out the initial bugs
Revision 2.0 99-6-30

Table of Contents

List of Tables

Preface

Printed documents tend to get outdated. If your paper copy has collected quite a lot of dust on the cover, you might want to check the following website for an update:

SGML for Windows NT (http://ourworld.compuserve.com/homepages/hoenicka_markus/ntsgml.html)

I. Preliminaries

Chapter 1. Introduction

What this is all about

SGML is a powerful metalanguage for writing markup texts. Regardless of whether you plan to develop your own SGML applications or simply plan to use existing ones, a few software components are required to edit and publish SGML documents. SGML sources are not meant to be read by humans directly (except the authors, of course), and publishing just means to convert the sources into a human-readable form of any kind. Commercial solutions are available for Windows NT, but they are expensive. Most Linux distributions contain a variety of SGML-related tools (I use and recommend Debian (http://www.debian.org), see the Debian SGMl/XML Howto (http://www.debian.org/~bortz/SGML-HOWTO/potato/howto.html)). Fortunately, all important SGML tools are available in Windows versions too, and moreover, they are available for free somewhere on the web. This tutorial describes my own setup of an useful selection of SGML tools on a Windows NT 4.0 box.

XML is a somewhat simplified version of SGML and has received a great deal of attention in the web business. XML is less intended as a replacement for HTML, although modern browsers can display XML files. Instead it is well suited for the exchange of data through the net and as a means to create HTML from XML just in time, a process that would be too slow for SGML documents. As XML is essentially a subset of SGML (with a few caveats), many SGML applications can process XML documents as well. To make best use of your XML documents, this tutorial presents a selection of dedicated XML software as well.

Who should read this tutorial?

This tutorial is clearly aimed at people who don’t have any experience with SGML or XML so far. However, by cross-reading it should also be helpful for the SGML geek who, for whatever reason, needs to quickly set up a free SGML/XML system on a Windows NT computer.

Contents in a nutshell

The next chapter contains a brief overview over the steps of writing and publishing SGML and XML documents. You will see what the components of the system are good for in the whole process. The whole matter is dealt with in three parts: The first part is about all those pieces of software that you need for both SGML and XML. The second and third part talk about software more or less specific to SGML and XML, respectively. The matter is subdivided into self-contained topics, which allow you to advance in small steps and to control your progress with simple examples and exercises.

> **Note:** You should browse the whole tutorial to get an understanding of what you actually need. If you use only SGML or only XML, you can leave the XML or the SGML part out, respectively. You may of course install everything which is suggested in the individual chapters, but you can always opt to leave things out. As an example, there are several ways to create printed output from SGML and

XML documents. The path through TeX is the most versatile, but requires some work to set up. If you are content with RTF output, you can just leave everything TeX-related out.

System Requirements

The software described below will run on Windows NT or Windows 95/98/ME. Windows 2000 and Windows XP are expected to work as well, but they were not tested. Installation on Windows 95/98/ME needs a few modifications of the installation procedure. This will be pointed out where necessary. The installed software occupies about 70 MB hard disk space. This may be considerably more if you use a FAT file system on a large harddisk instead of NTFS due to larger cluster sizes (consider partitioning the disk in this case). The memory size and CPU speed requirements are not extraordinary, I once ran the system on a Pentium 100 with 32 MB RAM and Windows NT 4.0 SP3 without problems. But I have to admit that especially OpenJade benefits noticeably from more RAM.

System Requirements II: Want Cygwin?

This tutorial actually comes in two flavours. The possibilities with the system as well as the installation instructions differ quite a bit depending on whether or not you want to use the Cygwin tools.

For starters, the Cygwin tools (http://www.cygwin.com) are a collection of GNU tools and a DLL which translates Unix-style system calls to native Win32 functions. This adds a POSIX layer on top of Win32 with a look and feel like any run-of-the-mill Unix or Linux. The most prominent change may be that you can use real shells like bash or csh instead of the limited Win32 shells (a.k.a. MS-DOS in Win9x). Shell scripts can simplify almost any complex task that you will be facing in the realm of SGML processing. There are Win32 users who consider the Cygwin Tools as “essential”. I strongly encourage you to use the Cygwin tools to simplify SGML processing. However, if you can’t stand the Unix way of computing, you will want to stick with the native Win32 version of this tutorial.

This is the version without the Cygwin Tools. If you need the other flavour of this tutorial, visit the SGML for NT homepage (http://ourworld.compuserve.com/homepages/hoenicka_markus/ntsgml.html).

Let us now praise free software

Setting up a free SGML editing and publishing system would not be possible without those programmers who release free software on the web. We should not take it for granted that we can obtain versatile, rock-solid, non-crashing software with full support through the Usenet or mailing lists, with the program sources and a license which entitles everybody to modify and improve it, at no cost. Many, many kudos to all those who make this possible.

Chapter 2. Overview: The components

This chapter briefly introduces the components that make up the complete system. The following chapters provide detailed installation instructions, and each chapter contains the exact download locations for the components.

Edit texts

Obviously we'll have to write or edit some kind of text if we want to create and publish SGML or XML documents. We will use the NT port of GNU Emacs. This is one of the most remarkable and versatile pieces of software, which can do anything you may think of except maybe preparing coffee. We will adapt and use it to edit SGML and XML source documents (using Lennart Staflin's PSGML mode), document type definitions (using Tony Graham's TDTD mode), DSSSL stylesheets, XSL stylesheets, and of course the editor's own configuration files. Moreover, we will use it as a kind of IDE by turning it into a frontend for the command-line utilities which do much of the work behind the scenes.

Validate SGML and XML documents

One of the strengths of SGML is the possibility to validate a document, that is to let a piece of software called a parser check the document whether or not it complies with a given DTD. We will use NSGMLS from James Clark's SP suite as a validating parser. This parser will work just fine for XML documents as well.

Publish SGML documents

There are several possiblities to publish (that is, convert to a human-readable form) SGML documents. The Document Style Semantics and Specification Language (DSSSL) is one of these methods to apply some kind of formatting rules to a SGML document. Jade, another piece of software written by James Clark, is the DSSSL application of choice for us. Jade is a modular application which can use several backends to create the kind of output that we need. Jade can create RTF (Rich Text Format, a sort of interchange format which is understood by most Windows and some Unix word processors), TeX (Donald Knuth's famous layout and typesetting engine), and MIF (FrameMaker format) output. It can also perform SGML-to-SGML transformations which can be used to create HTML output.

In the case of RTF, MIF, and HTML output, this is all you need. In the case of TeX output, we'll need a complete TeX system including the TeX interpreter itself, a previewer, and tools to convert the TeX output to PostScript or PDF format. We'll use a Win32 port of the TeX system called fpTeX. As PostScript is a useful cross-platform file format, we'll also need a software to preview these files. The GhostScript and GhostView utilities also allow to print PostScript files on a variety of non-PostScript printers.

Publish XML documents

While it would be possible to use a DSSSL engine for this purpose, XSL and XSLT are far more popular in the XML field. We will look at a few XML parsers that work as XSLT engines: Using XSLT stylesheets, they produce either a HTML representation of the document, or an intermediate XML document containing “Formatting Objects”. The latter can be further processed by formatting objects processors to create printable output.

Chapter 3. Some general remarks on installation procedures

This chapter briefly explains some general procedures which will be used throughout the installation. Please make sure you understand these few topics before you modify your system.

Administrator privileges

On a Windows NT/2000/XP box, all software should be installed with administrator privileges. This ensures that all users of the computer will be able to use the software without problems. Some packages won't install at all without administrator rights. Obviously this does not apply to Windows 95/98/ME, as with these operating systems every user is allowed to wreak havoc on the system (Windows ME has some recovery tools that try to save your butt *after* you screwed it up. They kindof sell you a car that crashes into the guardrails once per mile, but look: you get a toolbox and a rope for free! <Sorry - couldn't resist>).

Set environment variables

Environment variables are used by many programs to read in additional information at startup. E.g. the PATH environment variable tells the shell where to look for executables, thus allowing you to run many programs without having to specify the full path of the binary.

Windows NT and its successors know two types of environment variables: System variables can be changed only by the administrator and affect all users, whereas user variables affect only the user who set them for his own environment. Both are set in the System applet of the System settings group. Select the "Environment" tab in the pop-up dialog and click on the entry that you want to edit. The present value of the variable will be displayed in a separate edit field. Change the value or append new values to the semicolon-separated list. Click Set (clicking OK alone is not sufficient). To create a new environment variable, enter a new name into the provided edit field, and set a value as described above.

For Emacs as well as for command line windows, the environment variables are evaluated only once: when the window is opened first. If you change environment variables as described, they will have no effect on command line windows that are already open. So to see the effects of added or changed environment variables, you'll have to close and reopen any command line windows.

On Windows 95/98/ME, all environment variables are set in the `autoexec.bat` file with an entry like **set VARIABLE=VALUE**. You will have to reboot the computer to let the changes take effect.

Installation paths

Many of the tools that we'll use come from the Unix world. In Unix commands, a space is a token separator unless an expression is quoted. Unfortunately Microsoft was so proud of their long filenames in Windows95 (after every other OS had them since decades) that they didn't spend much time thinking how to use them properly. Thus spaces were allowed in paths, screwing up lots of applications from other

vendors. This general screw-up was neatly achieved by using `C:/Program Files` as the default folder for applications.

Please be warned that it is *not a good idea* to use paths with spaces for any of the tools used in this tutorial. Unless you positively want to shoot yourself in the foot, you should use something like `C:/Programs` as your applications folder and avoid all spaces in the subdirectories.

How to use archives

Some packages are provided as `.zip` files, which can be extracted with pkunzip or any Windows (un)zipper like WinZip. Be sure to keep the long filenames and the directory information which is provided in some archives. Others are provided as `.tar.gz` files. It is a common Unix format based on an uncompressed tape archive file (`.tar`) that is compressed in a second step (`.gz`). Recent versions of WinZip and possibly other tools support this format. When you open a `.tar.gz` file using WinZip, the software will either offer you to open the single `.tar` archive (simply accept this) or it will ask you to provide a filename for the single file it detects. Provide a filename with the suffix `.tar`, and the software will offer you to open this `.tar` archive. Then you can proceed as you would with any `.zip` archive. To simplify this whole procedure, you should make sure that your browser does not mangle the filename and replace all but the last dot with underscores in the "Save as" dialog when you download an archive.

How to use the code snippets

In the following chapters you will find a lot of code snippets that you will have to paste into configuration files. This is best accomplished by copying the code from the HTML version or the SGML source of this document. Rumors say that copying from PDF does not work reliably.

The sequence of the snippets that go into the `_emacs` file is not extremely critical. The most important thing is that the load-path is set correctly at the beginning of the file. If you walk through the tutorial, starting with a clean `_emacs` file, and add all relevant snippets to the end of your file, you should end up with a working configuration file. However, it is very unlikely that a little variety in the sequence would render your setup unusable, so feel free to experiment a little if you wish.

II. Common components

Chapter 4. Emacs

Emacs is the core component of the SGML/XML system. Due to its configurability and extensibility it will be the SGML and XML document editor and front-end to all related applications, just like an IDE for a programming language. Emacs is completely different from most other text editors in that it is basically a Lisp interpreter implemented in C and a whole bunch of Lisp files, which add most of the functionality.

NTEmacs is a port of GNU Emacs for the 32bit Windows platforms. The user interface retains most of the features of its Unix counterpart, so it takes some time to get used to it for the average MS Word user. However, there are quite a few tutorials and an extremely helpful reference card for the most important commands to get started quickly. There are some important things that you should know right now to understand what is happening during the installation below.

- Emacs is installed in a set of directories which will be automatically created when unpacking the archive. The `\bin` subdirectory holds the executables. The `\lisp` subdirectory holds all the Emacs Lisp code which turns Emacs into what it is. The `\site-lisp` subdirectory is a nice place for the configuration files `site-start.el` and `default.el` (see below).
- Emacs is configured by a set of startup files. As Windows NT is a multi-user system just like Unix, there are system-wide configuration files as well as user configuration files. When Emacs is started, it first reads a file called `site-start.el`. This file is meant to hold all the system-wide configuration data that users may override by their personal configuration files. It next reads the personal configuration file `_emacs` of the current user (NTEmacs also accepts the Unix-style `.emacs`, but you should be aware that files with a leading dot may be ignored by file-handling software like backup utilities, thus giving a new meaning to the leading dot as a hiding attribute on Unix systems). This file should contain personal modifications like color schemes, mail- or newsserver configuration or code to load specialized packages that other users on the system don't need. Finally it reads the file `default.el`, which can be used for code that users normally should not be allowed to override. If Emacs doesn't find any of these files, it starts up with sensible defaults which allow to access the basic functionality (and this is more than you'd expect).

  As the built-in Help⟶Customize feature affects only the `_emacs` file of the current user, we will make all configuration changes in this file throughout the whole tutorial. To turn this file into a system-wide configuration file, it is sufficient to move the contents or part of the contents from this file to your `site-lisp.el` or `default.el` afterwards.

- As Emacs reads the startup files only at startup, it is necessary to reload them after they were changed. The least elegant but safest way to do this is to restart Emacs.
- Due to its Unix heritage, Emacs accepts and sometimes expects path denominators in Unix style with forward slashes instead of the DOS-style backslashes. If you use backslashes in Emacs Lisp, be aware that this is a special character: the escape character. You'll need two consecutive backslashes to get one in the end, as in DOS-style paths.
- Also due to its Unix heritage, Emacs relies on the presence of a HOME environment variable which denotes the home directory of the current user. This may be any valid path that the user has write access to, including directories on network drives. When typing in paths, e.g. upon loading a file, the home directory can be abbreviated as ~.

- Emacs uses major and minor modes (defined in an `.el`/`.elc` file or in a set of such files) to add functionality which is not present by default.
- The Emacs Lisp source files (`.el`) can be byte-compiled (`.elc`) to increase the speed of execution. To minimize the archive size, the NTEmacs distribution contains only the byte-compiled versions. To modify the code or to learn from the code, you will have to download the sources separately.
- Instead of dialog boxes Emacs uses the minibuffer (the last line in the Emacs frame) to display messages and to enter commands. This command-line has many features in common with Bash (a widely used Unix shell), e.g. completion with the tab key and a sort of an argument history.

Gnuserv is a small utility which greatly simplifies the use of Emacs. If you associate file types with Emacs, each double-click on an associated file will start a new copy of the editor, thus wasting time and memory. Gnuserv allows to open such files in a running copy of Emacs, and it will create a new frame for the file if you wish so.

Although not directly related to Emacs, we will also install Ghostscript (a PostScript interpreter) and Ghostview (a graphical frontend for Ghostscript including a previewer) at this point. These tools are useful to preview PostScript documents and print them on non-PostScript printers. We need this feature in this and the following chapters to view or print useful help files. At a later stage you will be able to create PostScript documents from your SGML source documents.

Get the files

1. NTEmacs version 21.1 (ftp://ftp.gnu.org/gnu/windows/emacs/21.1/emacs-21.1-bin-i386.tar.gz). This contains the precompiled lisp files, but not the Lisp sources. If you want to edit the Lisp *source* code (or just want to learn from it), you can download the bigger full release (ftp://ftp.gnu.org/gnu/windows/emacs/21.1/emacs-21.1-fullbin-i386.tar.gz) instead.
2. Gnuserv (ftp://ftp.gnu.org/gnu/windows/emacs/contrib/gnuserv.zip)
3. Ghostscript version 7.00 (ftp://mirror.cs.wisc.edu/pub/mirrors/ghost/AFPL/gs700/gs700w32.exe)
4. Ghostview version 4.0 (ftp://mirror.cs.wisc.edu/pub/mirrors/ghost/ghostgum/gsv40w32.exe)

Emacs installation

1. Unpack the archive

   The emacs-21.1 archive contains directory information and long filenames, which both must be preserved when unpacking the files (see your unpacker help). Unpack the archive to your programs directory, e.g. `C:\Programs`. The unpacker will create the directory `\emacs-21.1` in your programs directory with the subdirectories `\bin`, `\etc`, `\info`, `\lisp`, and perhaps the empty subdirectories `\lock` and `\site-lisp`. To save some typing I suggest to rename the emacs directory e.g. to `\emacs211`.
2. Create the `\emacsen` subdirectory

In order to make future updates of NTEmacs as painless as possible, it is useful to keep all Elisp and other files that you will add to your basic NTEmacs installation in a separate directory. Therefore create a sibling of the emacs directory like `C:\Programs\emacsen`. The trick is that you can add and remove future versions of NTEmacs without touching your local add-ons. You can even run a new version in parallel to your existing version to see whether some of your local add-ons break. In this new subdirectory, create the subdirectories `\bin` and `\site-lisp`.

3. Adjust PATH

   Add the `\emacs211\bin` and `\emacsen\bin` directories to the PATH environment variable. This will let you start Emacs from the command line. In the control panel, select the "Environment" tab in the System dialog. Append the `\bin` directories, e.g. `c:\Programs\emacs211\bin;c:\Programs\emacsen\bin` to the PATH environment variable (this is a semicolon-separated list of directories) in the system variables panel (the upper one in WinNT, the lower one in Win2000) and press Set. Close the dialog by pressing Ok.

4. Create HOME

   Every user on your NT/2000/XP box has to define an environment variable called HOME. This variable will let Emacs find the personal `_emacs` file. Depending on your system, HOME should point either to a local or a network home directory. The variable is set via the control panel. Select the "Environment" tab in the System dialog and create HOME in the user variables panel (the lower one in NT, the upper one in Win2000 - by now you should see the pattern. I'll leave it as an extra bit of brain jogging for the XP users to find out where the panel is on their systems). Set its value to the appropriate directory, e.g. `c:\User\myself` and press Set. You can also use `%HOMEDRIVE%%HOMEPATH%` as a value for HOME instead. Close the dialog by pressing Ok.

   If you want to set up Emacs with a system-wide configuration file, set the HOME environment variable of your administrator account to an appropriate directory. After completing the whole SGML setup procedure from your administrator account, simply move the contents of the `_emacs` in your administrator home directory to one of the system-wide configuration files. This will be described at the end of this tutorial.

5. Set the load-path

   Now you'll have to make the first customizations to the `_emacs` file. The variable `load-path` tells Emacs where to look for Lisp files. The following chunk of code should be pretty much at the top of your `_emacs` file as other packages depend on `load-path`.

   ```
   ;; append some additional paths to load-path
   (setq load-path
          (nconc load-path (list "C:/Programs/emacsen/site-lisp"
         "C:/Programs/emacsen/site-lisp/psgml-1.2.4")))
   ```

   Adjust the paths as necessary. You could extend this list of paths for any additional Emacs Lisp packages that you install.

   > **Note:** This code adds the path of the PSGML package that we will install in the next chapter. Just keep in mind that we already took care of `load-path` at this point.

Gnuserv installation

1. Install `gnuserv.el`

   Extract `gnuserv.el` from the gnuserv archive and put it into your `\emacsen\site-lisp` directory, e.g. `C:\Programs\emacsen\site-lisp`. Start Emacs and byte-compile the file by issuing **Alt-x byte-compile-file [Return] c:/programs/emacsen/site-lisp/gnuserv.el**. This will create `gnuserv.elc`.

2. Modify your `_emacs`

   Add the following lines to your `_emacs`:

   ```
   (require 'gnuserv)
   (gnuserv-start)
   ```

3. Install the binaries

   Extract the `.exe` files from the gnuserv archive and put them into your `\emacsen\bin` directory, e.g. `C:\Programs\emacsen\bin`. Restart your Emacs.

4. Associate file types

   You can use the View⟶Options command in your Windows file explorer to associate file types with applications. Switch to the File types tab and associate whatever files you want with Emacs. Enter a command like **c:\programs\emacsen\bin\gnuclientw.exe "%1"** to the open action (enclosing the argument in "" helps to prevent problems with filenames that contain spaces). You can also create an equivalent link in your "Send to" folder which will allow you to open files with arbitrary (or no) extensions in Emacs with a right-click in the File Explorer.

Ghostscript/Ghostview installation

1. Unpack the archives

   Run the file `gs700w32.exe`. This will start a setup wizard. Enter a reasonable installation directory, e.g. `C:\Programs\Aladdin`. All other default settings should be ok. Press Next until the setup is complete.

   Run the file `gsv40w32.exe`. This will start another setup wizard. Enter an installation directory like `C:\Programs\Ghostgum` and press Next until the setup has finished. You will find a Ghostgum section in your Start menu which contains the links to the programs. Additionally, `.ps` and (if you selected this) `.pdf` files will be associated with Ghostview.

   > **Note:** If you intend to create `.pdf` files from your SGML sources, you probably want to use Ghostview as the default PDF viewer instead of Acrobat Reader. The latter locks all loaded files, so you can't change the sources and re-create the PDFs while they are displayed.

2. Configure Ghostview

Run GSView from the start menu. Another setup window will pop up and let you select the Ghostscript version that you want to use. The dialog box shows all installed versions of Ghostscript. Select the version 7.00 that you just installed and click Ok. The most recent versions of this program show a nag screen after startup which urges you to register. You should register if you wish to support further development of this tool, but the license explicitly does not require you to do so for noncommercial use.

Next, open the Media⟶Display Settings... dialog and set both *Text alpha* and *Graphics alpha* to 4. This will result in a more pleasant screen display (it does not affect print output).

3. Adjust PATH

   Add the Ghostview dir to the PATH environment variable. This lets you start Ghostview easily from the command line and simplifies some of the lisp code that we will add later to the startup files. In the control panel, select the "Environment" tab in the System dialog. Append the Ghostview directory, e.g. `c:\Programs\Ghostgum\gsview` to the PATH environment variable in the upper panel (system variables) and press Set. Close the dialog by pressing Ok.

4. Ghostview configuration

   Every NT user will have to perform Step 2 when she uses GSView for the first time. For the psgml-jade extension (see Install PSGML-Jade) to work properly, every user has to toggle off the Options⟶Save Last Directory menu command in GSView.

The first steps with Emacs

If you have never before worked with Emacs, it is no bad idea to go through the tutorial. Open it by using the Help⟶Emacs Tutorial menu command. You will learn the most important commands and shortcuts to use the general features of Emacs.

The Emacs package also contains a reference card. This 6-page document lists all important commands and shortcuts and is extremely useful until these commands are carved into your brain by constant use of the program. The Postscript document `refcard.ps` in the `\etc` subdirectory of your Emacs installation can be printed using GSView. Double-clicking the file `refcard.ps` should launch Ghostview and show the file after a few seconds. Click on the File⟶Print menu command to open the print dialog box. In addition to the usual printer selection box you will see a field called “Print Method”. Unless you call a PostScript printer your own, select “Windows GDI printer” and click OK.

Now you should be ready to play around a little. Open a few files which are lying around on your harddisk. Try e.g. C or C++ source files and see what happens (the menu will be extended with a C option containing useful commands to edit C source files). If you don’t have a C file around, start a new file from scratch: use the same command **C-c C-f** (that is: type c and then f while holding down the Control key) as for opening an existing file, but provide a location and name of the not yet existing file, e.g. `test.c`. Or simply drag and drop just any file on the Emacs window and see that it displays next to anything you try.

Further Reading

The Emacs FAQ for the Microsoft Windows port (http://www.gnu.org/software/emacs/windows/ntemacs.html), currently maintained by Steve Kemp, is a very comprehensive FAQ page for installing and using Emacs on the Windows platform.

Charles Curley maintains a page with installation instructions (http://w3.trib.com/~ccurley/emacs.html) for Emacs.

Additionally, an online manual (http://www.gnu.org/manual/emacs-20.7/emacs.html) (slightly outdated, though) is available. Manuals about Emacs Lisp are available for online viewing (http://www.gnu.org/manual/elisp-manual-20-2.5/elisp.html) or for download (http://paddington.ic.uva.nl/elisp-manual-20-2.5/).

Summary

Now, equipped with a working installation of GNU Emacs and Ghostview on your Windows computer, you should have learned a few things:

- Open and close files with Emacs, and finally quit Emacs
- Navigate in an Emacs buffer
- Mark, cut, copy, and paste regions of text
- Feel the ease of the command line completion
- View and print Postscript files with Ghostview

Chapter 5. PSGML and TDTD

The PSGML major mode allows the Emacs user to create and edit SGML and XML documents. PSGML is context-aware after parsing the document's document type definition (DTD). For example, commands like Markup⟶Insert Element offer only those elements or attributes which are allowed at the cursor position. This makes it much easier to write valid SGML documents, although it does not contain a validating parser. We will add validating parsers in later chapters of this tutorial.

As HTML is simply one (the best known, though) SGML application, it is tempting to use the PSGML mode for editing HTML files. We will therefore add some code which derives a HTML mode from PSGML.

TDTD is another Emacs major mode. This one turns Emacs into a DTD editor. This makes it easy to author and edit custom DTDs.

Get the files

1. PSGML-1.2.4 (http://www.sourceforge.net/projects/psgml/). Follow the link to the released files and download the archive file.
2. TDTD (http://www.menteith.com/tdtd/data/tdtd.zip)

Install PSGML

1. Unpack the PSGML archive

   Extract the PSGML archive to your common Emacs `site-lisp` directory, e.g. `C:\Programs\emacsen\site-lisp`. Be sure to keep the directory information and long filenames. This will create the subdirectory `\psgml-1.2.4`

2. Use onsgmls as an external validator

   The following code snippet lets PSGML use onsgmls from the OpenJade distribution (which we will install later) to validate SGML or XML files . Append this to your `_emacs` file:

   ```
   (setq sgml-validate-command "onsgmls -s %s %s")
   ```

3. Byte-compile the PSGML files

   To byte-compile all `.el` files in the `psgml-1.2.4` subdirectory, run the command (the "0" is a zero) **Ctrl-u 0 Alt-x byte-recompile-directory [Return] psgml-path**, where `psgml-path` is the path to your psgml-directory, e.g. `C:/Programs/emacsen/site-lisp/psgml-1.2.4`.

   > **Note:** If you get error messages like "cannot load file xyz" at this point, make sure that your Emacs load-path contains the PSGML directory. We already included this directory into the load-path when we installed Emacs.

4. Enable syntax coloring

   Edit `_emacs` to enable syntax coloring. Copy the following lines to your `_emacs` file (syntax coloring code adapted from David Megginson):

```
;; Turn on syntax coloring
(cond ((fboundp 'global-font-lock-mode)
;; Turn on font-lock in all modes that support it
(global-font-lock-mode t)
;; maximum colors
(setq font-lock-maximum-decoration t)))

;; load sgml-mode
(autoload 'sgml-mode "psgml" "Major mode to edit SGML files." t )

;; in sgml documents, parse dtd immediately to allow immediate
;; syntax coloring
(setq sgml-auto-activate-dtd t)

;; set the default SGML declaration. docbook.dcl should work for most DTDs
(setq sgml-declaration "c:/user/default/sgml/dtd/docbook41/docbook.dcl")

;; here we set the syntax color information for psgml
(setq-default sgml-set-face t)
;;
;; Faces.
;;
(make-face 'sgml-comment-face)
(make-face 'sgml-doctype-face)
(make-face 'sgml-end-tag-face)
(make-face 'sgml-entity-face)
(make-face 'sgml-ignored-face)
(make-face 'sgml-ms-end-face)
(make-face 'sgml-ms-start-face)
(make-face 'sgml-pi-face)
(make-face 'sgml-sgml-face)
(make-face 'sgml-short-ref-face)
(make-face 'sgml-start-tag-face)

(set-face-foreground 'sgml-comment-face "dark turquoise")
(set-face-foreground 'sgml-doctype-face "red")
(set-face-foreground 'sgml-end-tag-face "blue")
(set-face-foreground 'sgml-entity-face "magenta")
(set-face-foreground 'sgml-ignored-face "gray40")
(set-face-background 'sgml-ignored-face "gray60")
(set-face-foreground 'sgml-ms-end-face "green")
(set-face-foreground 'sgml-ms-start-face "yellow")
(set-face-foreground 'sgml-pi-face "lime green")
(set-face-foreground 'sgml-sgml-face "brown")
(set-face-foreground 'sgml-short-ref-face "deep sky blue")
(set-face-foreground 'sgml-start-tag-face "dark green")

(setq-default sgml-markup-faces
```

```
'((comment . sgml-comment-face)
(doctype . sgml-doctype-face)
(end-tag . sgml-end-tag-face)
(entity . sgml-entity-face)
(ignored . sgml-ignored-face)
(ms-end . sgml-ms-end-face)
(ms-start . sgml-ms-start-face)
(pi . sgml-pi-face)
(sgml . sgml-sgml-face)
(short-ref . sgml-short-ref-face)
(start-tag . sgml-start-tag-face)))

;; load xml-mode
(setq auto-mode-alist
(append (list (cons "\\.xml\\'" 'xml-mode))
auto-mode-alist))
(autoload 'xml-mode "psgml" nil t)
(setq sgml-xml-declaration "C:/Programs/OpenJade-1.3/pubtext/xml.dcl")
```

Use the Edit⟶Text Properties⟶Display Colors menu command in Emacs to see a list of supported color values if the colors in the above listing don't work on your system (or if you simply don't like them).

The XML declaration in the last line of the above listing is currently a dummy. We will update this if necessary when you install the necessary files.

5. Derive a HTML mode

   The following code derives a HTML mode from PSGML and was borrowed from the Debian Linux (http://www.debian.org) PSGML package. Insert the following lines into your `_emacs` file:

```
;; define html mode
(or (assoc "\\.html$" auto-mode-alist)
(setq auto-mode-alist (cons '("\\.html$" . sgml-html-mode)
auto-mode-alist)))
(or (assoc "\\.htm$" auto-mode-alist)
(setq auto-mode-alist (cons '("\\.htm$" . sgml-html-mode)
auto-mode-alist)))

(defun sgml-html-mode ()
"This version of html mode is just a wrapper around sgml mode."
(interactive)
(sgml-mode)
(make-local-variable 'sgml-declaration)
(make-local-variable 'sgml-default-doctype-name)
(setq
sgml-default-doctype-name    "html"
sgml-declaration             "c:/user/default/sgml/dtd/html/html.dcl"

sgml-always-quote-attributes t
sgml-indent-step             2
sgml-indent-data             t
sgml-minimize-attributes     nil
```

```
sgml-omittag                t
sgml-shorttag               t
)
)

(setq-default sgml-indent-data t)
(setq
sgml-always-quote-attributes   t
sgml-auto-insert-required-elements t
sgml-auto-activate-dtd         t
sgml-indent-data               t
sgml-indent-step               2
sgml-minimize-attributes       nil
sgml-omittag                   nil
sgml-shorttag                  nil
)
```

The path in the variable sgml-declaration in the listing above is again a dummy at the moment. We will adjust this path as soon as all necessary files are installed.

Install TDTD

1. Extract the files

   Install the tdtd mode, using the tdtd archive. Unzip the files `tdtd.el` and `tdtd-font.el` into your local site-lisp directory, e.g. `C:\Programs\emacsen\site-lisp`.

2. Byte-compile the files

   Start Emacs and byte-compile the file `tdtd.el` by typing **Alt-x byte-compile-file [Return] site-lisp-path/tdtd.el [Return]**, where site-lisp-path is your site-lisp directory. Use the same procedure to byte-compile `tdtd-font.el`.

3. Modify your _emacs

   Copy the following lines into your _emacs file:

   ```
   ;; Start DTD mode for editing SGML-DTDs
   (autoload 'dtd-mode "tdtd" "Major mode for SGML and XML DTDs.")
   (autoload 'dtd-etags "tdtd"
   "Execute etags on FILESPEC and match on DTD-specific regular expressions."
   t)
   (autoload 'dtd-grep "tdtd" "Grep for PATTERN in files matching FILESPEC." t)

   ;; Turn on font lock when in DTD mode
   (add-hook 'dtd-mode-hooks
   'turn-on-font-lock)

   (setq auto-mode-alist
   (append
   ```

```
(list
'("\\.dcl$" . dtd-mode)
'("\\.dec$" . dtd-mode)
'("\\.dtd$" . dtd-mode)
'("\\.ele$" . dtd-mode)
'("\\.ent$" . dtd-mode)
'("\\.mod$" . dtd-mode))
auto-mode-alist))

;; the regexp for NTEmacs etags
(setq dtd-etags-regex-option
"-regex=\'/<!\\(ELEMENT\\|ENTITY[ \\t]+%\\|NOTATION\\|ATTLIST\\)[ \\t]+\\([^ \\t]+\\)/\\
```

The first steps with PSGML

Now you should have a look at an SGML document with PSGML. We'll start with a very simple example provided by Paul Prescod's DSSSL introduction (see Further Reading). This example does not use an external DTD. The necessary files to resolve links to external DTDs will be added later. There is no need to understand the syntax of the DTD at the moment, simply type it or copy it to your document.

> **Note:** If you prefer to start with an XML document right away, you may peek ahead to the chapter about xslide. There you'll find an XML version of the SGML document described here.

Restart Emacs to apply the latest changes to your `_emacs` and open a new SGML document: Type **C-x C-f ~/test.sgml** to create the document `test.sgml` in your home directory (you may as well type the full path of a file in any other convenient directory). Emacs should load the SGML mode (check the modeline close to the bottom of the Emacs window). Now type or paste the following DTD into the new buffer:

```
<!DOCTYPE HTMLLite [
<!ELEMENT HTMLLite O O (H1|P)* >
<!ELEMENT (H1|P) - - (#PCDATA|EM|STRONG)* >
<!ELEMENT (EM|STRONG) - - (#PCDATA) >
]>
```

As you just have created the DTD, PSGML is not yet aware of it (when you load an existing SGML file, PSGML will automatically parse the DTD due to a setting in your `_emacs` file). Press **C-c C-p** to parse the DTD now. Check your DTD again if PSGML issues an error.

Now place the cursor at the end of the buffer and start to append the contents of the document to the prolog. Use the menu command Markup⟶Insert Element to enter the first element. You will notice that PSGML offers only one possible element to insert: HTMLLite. Accept this and insert another element right where you are. This time you will see that you can either insert a H1 or a P element. You can enter some text between the H1 or P start and end tags. Try to insert another element when the cursor

is inside the text in a H1 or P element. This will allow you to insert either a STRONG or an EM element, which again can hold some text. Try also the keyboard shortcut for inserting an element, **C-c C-e**. You will be prompted for the name of an element. Either enter the full name of an element, the first letters and then the tab key, or simply press the tab key twice to see a list of possible tags at the given location. Note that you don't have to enter hard returns or to manually indent, PSGML takes care of this.

When you're done, save your document by pressing **C-c C-s**. A valid HTMLLite document may look like this:

```
<htmllite>
<h1>Character of Constantine</h1>
<p>The person, as well as the mind, of Constantine had been enriched by nature
with her choicest endowments.</p>
<p>His stature was <em>lofty</em>, his countenance <em>majestic</em>, his
deportment <em>graceful</em>; his strength and activity were displayed in
every manly exercise, and, from his earliest youth to a very advanced season
of life, he preserved the vigour of his constitution by a
<strong>strict</strong> adherence to the domestic virtues of chastity and
temperance.</p>
</htmllite>
```

You should also have noticed that PSGML performs syntax-highlighting: Start/end tags and other language-related constructs are shown in different colors to increase readability. Do not delete your test document yet, because you may wish to reuse it in the next chapter.

Further Reading

Norman Walsh's *DocBook: The Definitive Guide* is available as a book at O'Reilly (http://www.ora.com) and online (http://www.nwalsh.com/docbook/defguide/index.html). The introductory chapter contains an excellent overview about the concept of SGML.

The PSGML package contains a handbook in Postscript format which you can print using Ghostview. This is an valuable reference which explains the commands and the variables which you may use for further customization.

Bob DuCharme has published online a chapter of his SGML book which explains how to write and modify SGML files using Emacs and PSGML. These 100 pages called "Editing SGML documents with the Emacs text editor (http://www.snee.com/bob/sgmlfree/emcspsgm.zip)" are an excellent tutorial to learn and understand PSGML thoroughly.

The TDTD archive contains `tutorial.txt` which covers the editing of DTDs with TDTD.

Summary

Now you should be able to:

- Write simple SGML documents
- Insert elements or start/end tags

Chapter 6. TeX

TeX is Donald Knuth's famous typesetting software that was born from the inability of the computer systems of the early '80 to output anything that comes close to a decent mathematical text. The typesetting algorithms are derived from long-standing rules in the book printing business, and the default output looks quite different (read: more pleasant) than any word processor output. Of course *everything* is configurable. The primary printable output (a DVI file) is *device-independent*, i.e. the printed output will be the same regardless of the operating system and the printer that you may use (remember that Windows word processors shift line- and pagebreaks back and forth if you just dare to move from a 300dpi to a 600dpi printer?). Furthermore the system comes with utilities to create Postscript files (the standard print format on Unix systems) and PDF files (the inofficial standard print format of the web).

TeX is big, completely different from a word processor, and at times unwieldy for the uninitiated. If you just want to print something somehow, the SGML->RTF conversion or the XML->PDF conversion may be sufficient. If you look for highest-quality typesetting and easy conversion to platform-independent formats, you should seriously consider installing TeX.

If you have never before worked with TeX or LaTeX, there are a few things you should know:

- TeX is a markup language just like SGML, but it lacks the strict separation of content and formatting instructions.
- The TeX system resides in a directory tree which is usually called `\texmf`. The subdirectories contain all the binaries, configuration files, fonts, and macros. Additionally, a `\texmf-local` tree can be used. This tree holds modified configuration files and format files. This prevents them from being overwritten if you update the TeX installation.
- Unlike a word processor, TeX does not have a graphical user interface. You type the TeX source text in whatever editor you prefer. In a later section of this tutorial we will also install a TeX major mode which turns Emacs into a nice TeX editor. In the case of SGML documents, the intermediate TeX file will be created by OpenJade's TeX backend.
- The TeX command-line application interprets the markup instructions of the source document and creates a `.dvi` (device independent) output file. This file can be further converted to a Postscript file to view or print it with Ghostview.
- TeX uses auxiliary files to create tables of contents, lists of figures or tables, and other kinds of cross-referencing. Therefore you may need up to three passes to get all references right (TeX tells you whether or not you need an additional pass).
- TeX uses the MetaFont application and font descriptions to generate bitmap fonts on the fly for the requested font sizes and styles. These bitmap font files remain on the system unless you manually remove them. Therefore the necessary MetaFont runs will become less frequent or stop after a while.
- Everything that you need for TeX and then some is available at CTAN (Comprehensive TeX Archive Network). This is a group of servers that provide a plethora of TeX-related files in an identical directory layout. A list of available mirrors can be found here (http://www.dante.de/software/ctan/CTAN.sites). A HTML-based search is available at www.dante.de (http://www.dante.de/cgi-bin/ctan-index).
- The TeX binaries have a *very* long history and carry a lot of legacy code with them. Among this is the use of fixed size buffers for various internal things. The size of these buffers is defined in a

configuration file (`web2c/texmf.cnf`). Although we start with pretty generous settings it is possible to hit the limit with very large or very complex SGML or XML files. The error message will tell you which variable was limiting. You should then increase the value in the configuration file and rebuild the format files with the command **texconfig init** as shown below.

The probably best known TeX distribution is teTeX. fpTeX is a Win32 version of TeX based on teTeX.

Our customized fpTeX installation contains two macro sets which are necessary to create printable output from SGML or XML documents through TeX. They will be automatically configured so we don't have any extra work to do. Just keep in mind that they are there. JadeTeX is a macro set that complements Jade's TeX backend. xmlTeX and PassiveTeX do a similar job for the FO output of XSLT engines.

Get the files

1. TeXSetup.exe (ftp://ftp.dante.de/pub/fptex/current/TeXSetup.exe). This is the install program to bootstrap your fpTeX installation.
2. ulem.sty (ftp://ftp.dante.de/tex-archive/macros/latex/contrib/other/misc/ulem.sty) and url.sty (ftp://ftp.dante.de/tex-archive/macros/latex/contrib/other/misc/url.sty). These are two style files needed by JadeTeX that apparently are not in any of the fpTeX packages.
3. Visit a CTAN server, e.g. ftp.dante.de (ftp://ftp.dante.de), to retrieve some additional TeX-related files which are not part of the TeX distribution. *This is not necessary in most cases*, but if you deal a lot with foreign languages or with functional programming, you may want to have the following packages (I don't provide full links here; please use a CTAN mirror and visit the given directories):
   - tipa (International Phonetic Alphabet): CTAN:fonts/tipa
   - mmasym: Virtual TeX fonts for use with Mathematica 3.0 PostScript fonts: CTAN:fonts/psfonts/Mathematica 3.0

Install TeX

1. Create the directory structure

   In order to do the installation in one fell swoop it is prudent to create the directory structure beforehand, at least partially. The reason is that we need to provide two additional style files before the fpTeX installer attempts to build the format files. Create a suitable installation directory hierarchy like `C:\Programs\TeXLive\texmf-local\tex\latex`. Please do not use paths with spaces unless you *want* trouble at some point. You should have a fair amount of free space (up to 300 Mb) on that drive or partition. Copy the two files `ulem.sty` and `url.sty` into the new `latex` subdirectory.

2. Run TeXSetup

   Run the file `TeXSetup.exe`. This is a wizard-style application that collects all necessary information from you.

- Set the fpTeX installation directory according to what you previously created, that is `C:\Programs\TeXLive` for the example above. fpTeX will be installed in subdirectories (`texmf` and `texmf-local`, among others) of this directory.
- On the next page, select “download from internet” and “direct connection”. Clicking next will start the download process for the current package list which will take only a minute or two.
- Now you should see the package selection page. The essential packages are already selected. For our purposes you should add a few more. These are:
  - tex-extrabin
  - tex-fontbin
  - tex-fontsextra
  - tex-htmlxml
  - tex-mathextra
  - tex-psfonts
  - tex-psutils
  - one or more of the tex-langXYZ packages if you need support for languages other than the standard set.
- Start the file transfer. A cup of coffee may be warranted at this point. At the end of the download process, the installer will initialize the installation, create the filename databases, and build the format files. Check the log file for any suspicious error messages.

3. Configure dvips

   You’ll probably have to change some settings for dvips for your local site. The settings are stored in the file config.ps in the `\Texmf\dvips\config` directory. The following options might need a change:

   - The line starting with M should be modified according to the make and model of the printer that you use. The file `\metafont\misc\modes.mf` contains a list of possible values. Remember that a printer which is not in the list may be happy with the mode of a related printer, e.g. most recent HP LaserJet printers work just fine with the ljet4 setting, and many other cheap non-HP laser printers work with the ljet2 setting.
   - The line starting with D specifies the resolution of your printer. This always means the physical resolution, not what some resolution enhancement technology claims to make of the latter.
   - The line starting with O (the capital letter, not zero) specifies the horizontal and vertical printer offsets. These are printer-specific values which can be used to adjust the printout on the paper. The easiest way to determine these values is to use the file `testpage.tex`, which you will find in the `\texmf\tex\latex\base` directory. Open a command-line window and change into this directory. Type **latex testpage.tex** and answer the questions that appear on your screen. A `testpage.dvi` will be created. Use **dvips -o testpage.ps testpage.dvi** to convert this to the PostScript file `testpage.ps`. Now print this file on your printer using GhostView. You will see a box which should be evenly spaced with a 1 inch margin on all sides. The rulers help you to calculate which offsets (if at all) have to be specified in `config.ps` to adjust the printout

correctly. The offsets can be specified as *cm*, as *in*, or as *pt*. In addition, these rulers show you the printer-specific unprintable area on the edges.

The first steps with TeX

Although there is no need to learn how to write TeX documents in order to turn our SGML or XML documents into printable output, we should take the time to perform a small test with our TeX installation at this point. Without TeX being set up properly, the TeX backend will be useless anyway. Even if you don't know anything about TeX, simply follow the instructions below.

Start a new TeX file in your Emacs by typing **C-x C-f textest.tex**. Enter or paste the following lines:

```
\documentclass[letter]{article}

\begin{document}

\section{X, 34}
For one bitten by true doctrines even the briefest and most familiar saying
is reminder enough to dispel sorrow and fear, for instance:

\begin{verse}
Like as the generation of leaves, even such are the children of men.\\
The wind scatters them on the face of the ground, but others the woodland\\
Brings forth again in its strength and they shoot in the season of spring;\\
Like to them are the children of men, one waxes, another is waning.
\end{verse}


\end{document}
```

Now save the file by typing **C-x C-s**. Open a command-line window and change to the directory where you saved your `textest.tex` file. Now enter the command:

```
C:\user\myself>latex textest.tex
```

This should create the files `textest.aux`, `textest.log`, and `textest.dvi`, which are an auxiliary file, a log file, and the formatted document, respectively. View the output by typing:

```
C:\user\myself>dvips -o textest.ps textest.dvi
C:\user\myself>gsview32 textest.ps &
```

This should open Ghostview and load the output file. You should see a nicely formatted poem now.

Further Reading

The fpTeX distribution contains an extensive manual in HTML format. See also the documentation files in the `texmf/doc/programs` directory of your TeX installation. For further TeX- and LaTeX-related

information see the LaTeX cookbook (http://star-www.rl.ac.uk/docs/sc9.htx/sc9.html) and the Not so short Introduction to LaTeX2e (ftp://ftp.dante.de/pub/tex/info/lshort/english/lshort.pdf).

Summary

In this chapter you should have learned the following things:

- Writing a TeX document is not that hard after all, even if you will not have to do this in the context of our further SGML or XML endeavours.
- Running a TeX file through the tex compiler

III. SGML processing

Putting together an SGML processing system is fairly straightforward in the sense that there is not so much choice. DSSSL is the predominant stylesheet language for SGML transformation, and Jade is the only sufficiently complete free DSSSL engine (OpenJade is a newer version of Jade with a few additions). Our task is also simplified by the fact that this DSSSL engine has several backends which create both HTML and printable output formats.

Chapter 7. OpenJade and onsgmls

Now we are able to create SGML source files with Emacs. PSGML parses the markup and prevents some, but not all markup errors. Still worse, our documents look as ugly as HTML source files which nobody will enjoy to read. So on the one hand we need an external validating parser to get validated SGML documents. On the other hand, there must be a way to create human-readable, formatted output from our sources. The SP suite written by James Clark contains all necessary tools. NSGMLS is the validating parser to check our documents. Jade (James' DSSSL engine) is an implementation of the DSSSL style language which takes a SGML document (which holds the content) and a stylesheet (which holds the formatting instructions) as input to either create printable output (using the RTF, TeX, or MIF backends) or transform it into another SGML document, e.g. HTML (using the SGML backend).

As James Clark does not have any plans for further development of Jade currently, a group of volunteers maintains newer versions of Jade. To distinguish them from the older version, the tools in the new releases were renamed to openjade and onsgmls etc., and the whole suite is maintained as two packages, OpenJade and OpenSP.

> **Note:** As XML is essentially a subset of SGML, it is perfectly possible to use the tools described in this chapter for XML files as well. It is not uncommon to use onsgmls to validate XML files against a DTD.

Get the files

1. OpenJade/OpenSP 1.3 Win32 binaries (http://sourceforge.net/projects/openjade). Follow the link to "OpenJade 1.3 Win32 Bins"

Install OpenJade and the OpenSP suite

1. Extract the files

   Extract the OpenJade archive into a suitable directory for installation, e.g. `c:\Programs\OpenJade-1.3`.

2. Adjust PATH

   Change the PATH system environment variable to include the OpenJade `bin` directory. This is done via the system settings. Select the "Environment" tab in the System dialog, click on the PATH entry in the upper pane (System variables), and append the path, e.g. `C:\Programs\OpenJade-1.3\bin`, to the semicolon-separated list. Click Set and do not yet close the dialog.

3. Create SGML_CATALOG_FILES

   Create a SGML_CATALOG_FILES system environment variable. This variable tells the SP applications where to look for catalog files. These catalog files are used to resolve public identifiers to local filenames. In the Environment tab of the system dialog enter SGML_CATALOG_FILES as a new variable name in the upper pane. The value of this variable is a semicolon-separated list of catalog files. Now there is only one catalog to be included, which is provided by the Jade package. The current directory should be included as well (assuming that the catalog file is called `catalog`, which is often the case) as well as the SP catalog file. If you put the catalog file of the current directory on the first position of the list, you can easily override any system catalogs by simply providing a catalog file in your document's directory. You will later modify this variable, when you install the HTML and DocBook DTDs. Be aware that this variable is not a list of directories that contain catalog files, but rather a list of the paths of individual catalog files. Set the initial value of the environment variable e.g. as follows:

   ```
   .\catalog;C:\Programs\OpenJade-1.3\dsssl\catalog
   ```

   Individual users can add private catalog files by defining a user environment variable SGML_CATALOG_FILES which holds the full paths of these catalog files (don't forget to include "%SGML_CATALOG_FILES%" in the user's lists).

The first steps with OpenJade and onsgmls

In this section we will have a brief look at the two tools openjade and onsgmls. At the moment we will use them by typing commands on the command line. This will give you some understanding about these tools, but keep in mind that you will be able to run these tools with a few mouseclicks from within Emacs at a later stage of this tutorial.

At first, we will check whether our `test.sgml` is a valid SGML file. As onsgmls is a command-line application, first open a NT command-line window. Change to the directory where your `test.sgml` was saved to and type the following command:

```
c:\user\myself> onsgmls -s test.sgml
```

> **Note:** The above command is the simplest possible case that happens to work for us because our DTD is very simple. In real life, you will have to provide a SGML declaration as the first non-option argument on the command line (the second argument is then the name of the file you want to process). Processing XML files requires additional command line switches besides a correct SGML declaration. Thus for a DocBook SGML document you would run:
>
> ```
> c:\user\myself> onsgmls -s \path\to\docbook.dcl test.sgml
> ```
>
> And for some XML document you'd have to type:
>
> ```
> c:\user\myself> onsgmls -wxml -s \path\to\xml.dcl test.xml
> ```
>
> The `-wxml` switch gives you extended warnings for XML documents, while the `-s` switch suppresses the output of the parsed document contents.

This will parse the document and list any errors that it contains. Don't worry if onsgmls produces no output on the screen. This Unix-style brevity just means that there were no errors. To see an error message, change a tag in your `test.sgml` source file, e.g. change a h1 tag to read h and run onsgmls again.

Before we can try to process our test document with OpenJade, we need a stylesheet for our DTD. We will use a very simple stylesheet to create printable output from the test file that you created in the previous chapter. The stylesheet is borrowed from Paul Prescod, just like the DTD. Open a new file with the name `test.dsl` in the same directory as your `test.sgml` and type in or paste the following lines:

```
<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN" >
<style-sheet>
<style-specification>
<style-specification-body>

  (element HTMLlite (make simple-page-sequence))

  (element H1
  (make paragraph
  font-family-name: "Times New Roman"
  font-weight: 'bold
  font-size: 20pt
  line-spacing: 22pt
  space-before: 15pt
  space-after: 10pt
  start-indent: 6pt
  first-line-start-indent: -6pt
  quadding: 'center
  keep-with-next?: #t))

  (element P
  (make paragraph
  font-family-name: "Times New Roman"
  font-size: 12pt
  line-spacing: 13.2pt
  space-before: 6pt
  start-indent: 6pt
  quadding: 'start))

  (element EM
  (make sequence
  font-posture: 'italic))

  (element STRONG
  (make sequence
  font-weight: 'bold))
</style-specification-body>
</style-specification>
</style-sheet>
```

openjade is a command-line tool, just like onsgmls. Open a command-line window from your start menu. Change into the directory which contains your test SGML source and your stylesheet. There are some command-line options (see the file `index.html` in the `OpenJade\jadedoc` subdirectory), but all we need to know right now is the -t option to select the backend and the -d option to specify the stylesheet. We want to create a RTF file from our document, so the command is:

```
c:\user\myself> openjade -t rtf -d test.dsl test.sgml
```

openjade should silently create the file `test.rtf` in the current directory. Try to open this document with any RTF-capable word processor (WordPad is sufficient for such a simple file).

Feel free to modify the stylesheet to better understand the effect of the style specifications. Change the font sizes or the fonts themselves, or change the line spacing. Use the additional Jade command line option **-o output_file** to provide a different output filename, e.g. `test1.rtf`, so you can easily compare the original output and the output created with your modified stylesheet.

With just a little more effort, we can also utilize the TeX backend and our TeX installation to create printable output:

```
C:\user\myself>openjade -t tex -d test.dsl test.sgml
```

This will create the file `test.tex`. This is a LaTeX file which needs some special macros (the JadeTeX stuff) to be interpreted correctly by TeX. So the command line is:

```
C:\user\myself>jadetex test.tex
```

This should create the standard set of `.aux`, `.log`, and `.dvi` files.

Create and view a PostScript document from your `test.dvi`. To this end, type the command:

```
C:\user\myself>dvips -o test.ps test.dvi
```

and view the resulting file with:

```
C:\user\myself>gsview32 test.ps
```

Alternatively, try to directly create a PDF document by typing:

```
C:\user\myself>pdfjadetex test.tex
```

View the PDF file by typing:

```
C:\user\myself>gsview32 test.pdf
```

Further Reading

The OpenJade package contains a set of `.html` help files in the `OpenJade\jadedoc` subdirectory which explain the usage and the command-line options of all tools. Henry Thompson prepared a digest (ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/digest.htm) of the somewhat dry DSSSL specification (ftp://ftp.ornl.gov/pub/sgml/WG8/DSSSL/dsssl96b.pdf). Now it’s also a good time to look at some

introductory texts describing the concept and usage of DSSSL. A good place to start is provided by Netfolder (http://www.netfolder.com/DSSSL/index.html).

Paul Prescod has compiled a very useful Introduction to DSSSL (http://www.prescod.net/).

Summary

You should have learned the following things now:

- Validate a SGML document using onsgmls
- Create a RTF version from a simple SGML document and its stylesheet using openjade.

Chapter 8. IDE helpers

In this chapter we'll get rid of the need to type strange commands into our command lines and instead use some Emacs Lisp packages to turn Emacs into an OpenJade and TeX frontend. PSGML-Jade turns Emacs into an OpenJade frontend, providing an alternative to the command-line interface of the publishing tools. AucTeX is a package which turns Emacs into a TeX editor. Strictly speaking, AucTeX is not necessary for the SGML system proper, but it will help you to insert some manual pagebreaks into OpenJade's TeX backend output if this should be necessary. Finally, PSGML-DSSSL is an Emacs-Lisp file which allows you to create a skeleton DSSSL stylesheet based upon your current DTD.

Get the files

1. psgml-jade.el (http://ourworld.compuserve.com/homepages/hoenicka_markus/psgml-jade.el)

   Note: This file is not identical with a file of the same name on the OpenJade project page at sourceforge (http://www.sourceforge.net/projects/openjade). Their version seems to be derived from an older version of `psgml-jade.el` and does not support Windows. I'll try to consolidate these versions as soon as time permits.

2. psgml-dsssl (http://www.megginson.com/Software/psgml-dsssl.el)
3. AucTeX (ftp://ftp.dina.kvl.dk/pub/Staff/Per.Abrahamsen/auctex/auctex.tar.gz)

Install PSGML-Jade

1. Copy and byte-compile the file

   Copy the file `psgml-jade.el` into your common site-lisp directory, e.g. `C:\Programs\emacsen\site-lisp`.

   Byte-compile the file `psgml-jade.el` by typing **Alt-x byte-compile-file [Return] site-lisp-path\psgml-jade.el [Return]**, where site-lisp-path is your site-lisp directory. This creates `psgml-jade.elc`.

2. Modify your `_emacs`

   Insert the following block of code into your `_emacs`:

```
;; load psgml-jade extension
(setq
  sgml-command-list
  (list
```

```
   (list "Jade" "c:\\Programs\\OpenJade-1.3\\openjade -c%catalogs -t%backend -
d%stylesheet %file"
  'sgml-run-command t
  '(("jade:\\(.*\\):\\(.*\\):\\(.*\\):E:" 1 2 3)))
   (list "JadeTeX" "jadetex %tex"
  'sgml-run-command nil)
   (list "JadeTeX PDF" "pdfjadetex %tex"
  'sgml-run-command t)
   (list "dvips" "dvips -o %ps %dvi"
  'sgml-run-command nil)
   (list "View dvi" "windvi %dvi"
  'sgml-run-background t)
   (list "View PDF" "gsview32 %pdf"
  'sgml-run-command nil)
   (list "View ps" "gsview32 %ps"
  'sgml-run-command nil))
)

(setq sgml-sgml-file-extension "sgml")

(setq sgml-dsssl-file-extension "dsl")

(setq sgml-expand-list
  (list
   (list "%file" 'file nil)  ; the current file as is
   (list "%sgml" 'file sgml-sgml-file-extension) ;   with given extension
   (list "%tex" 'file "tex")  ;   dito
   (list "%dvi" 'file "dvi")  ;   dito
   (list "%pdf" 'file "pdf")  ;   dito
   (list "%ps" 'file "ps")  ;   dito
   (list "%dsssl" 'file sgml-dsssl-file-extension) ;   dito
   (list "%dir" 'file nil t)  ; the directory part
   (list "%stylesheet" 'sgml-dsssl-spec) ; the specified style sheet
   (list "%backend" 'sgml-jade-backend) ; the selected backend
   (list "%catalogs" 'sgml-dsssl-catalogs 'sgml-catalog-files 'sgml-local-catalogs)
     ; the catalogs listed in sgml-catalog-files and sgml-local-catalogs.
   )
)

(setq sgml-shell "C:/Programs/emacs211/bin/cmdproxy.exe")

(add-hook 'sgml-mode-hook '(lambda () (require 'psgml-jade)))
```

Adjust the path in the line defining **sgml-shell** to point to your Emacs `/bin` directory. Adjust the path to the OpenJade binary if necessary. Save your `_emacs`.

Install PSGML-DSSSL

Copy the file `psgml-dsssl.el` into your common site-lisp directory, e.g. `C:\Programs\emacsen\site-lisp`. Start Emacs and byte-compile the file `psgml-dsssl.el` by typing **Alt-x byte-compile-file [Return] site-lisp-path\psgml-dsssl.el [Return]**, where site-lisp-path is your site-lisp directory. This creates `psgml-dsssl.elc`.

Copy the following lines into your `_emacs` file:

```
;; load dsssl support
(autoload 'sgml-dsssl-make-spec "psgml-dsssl" nil t)
```

Install AucTeX

1. Extract the files

   Extract the Auctex archive to your common `site-lisp` directory, e.g. `C:\Programs\emacsen\site-lisp`. Be sure to keep directory information and long filenames. A new subdirectory will be created for the Auctex files.

2. Move tex-site.el

   Locate the file `tex-site.el` in the Auctex subdirectory that was created in the previous step and move it one level up to the `\site-lisp` directory in your common Emacs directory.

3. Byte-compile the files

   Start Emacs and byte-compile the Auctex subdir by typing (the 0 is a zero) **Ctrl-u 0 Alt-x byte-recompile-directory [Return] C:/Programs/emacsen/site-lisp/auctex-11.10 [Return]**. Some more files are in the `style` subdirectory, so you should also run **Ctrl-u 0 Alt-x byte-recompile-directory [Return] C:/Programs/emacsen/site-lisp/auctex-11.10/style [Return]**.

4. Modify tex-site.el

   Edit `tex-site.el` in the `\emacsen\site-lisp` subdirectory. Locate the line:

   ```
   (defvar TeX-lisp-directory "@AUCDIR"
   "*The directory where the AUC TeX lisp files are located.")
   ```

   and replace the string *@AUCDIR* with your Auctex directory, e.g. `C:/Programs/emacsen/site-lisp/auctex-11.10`.

   Immediately after these lines, insert the following code (modified from `tex.el`):

   ```
   ;; Change this to point to the place where the TeX macros are stored
   ;; at yourt site.
   (defcustom TeX-macro-global '("c:/Programs/TeXLive/texmf/tex/")
   ```

```
"Directories containing the sites TeX macro files and style files.
The directory names *must* end with a slash."
:group 'TeX-file
:type '(repeat (directory :format "%v")))

;; The 'TeX-command-list' (pull-down menu at the top of emacs appearing when
;; emacs is in TeX major mode) consists of the options below.
;; Invoking 'C-c C-c' in a TeX major mode will run the "LaTeX" command
;; of the command list. (After compiling, errors can be retrieved by
;; invoking 'C-c `' (Control-c accent-gr\`ave).
;; If no errors occur and if all cross-references are known, a second
;; 'C-c C-c' will run the 'View' command of the list.
(defvar TeX-command-list
(list (list "TeX" "tex \\nonstopmode\\input{%t}" 'TeX-run-TeX nil t)
(list "LaTeX" "latex \\nonstopmode\\input{%t}"
'TeX-run-LaTeX nil t)
(list "View DVI" "windvi.exe %d"
'TeX-run-command nil t)
(list "PDFLaTeX" "pdflatex \\nonstopmode\\input{%t}"
'TeX-run-LaTeX nil t)
(list "View PDF" "gsview32.exe %a"
'TeX-run-command nil t)
(list "dviPS" "dvips %d -o %f"
'TeX-run-command nil t)
(list "View PostScript" "gsview32.exe %f"
'TeX-run-command nil t)
(list "BibTeX" "bibtex %s" 'TeX-run-BibTeX nil nil)
(list "Index" "makeindex %s" 'TeX-run-command nil t)
(list "Check" "lacheck %s" 'TeX-run-compile nil t)
(list "Other" "" 'TeX-run-command t t)))

(setq TeX-default-mode 'LaTeX-mode)
(setq LaTeX-command-style '(("." "latex -src-specials")))
(setq TeX-view-style '(("^a5$" "windvi %d -paper a5")
("^landscape$" "windvi %d -paper a4r -s 4")
("^epsf$" "gsview32 %f")
("." "windvi -single %d")))
```

On line 3 of the inserted code the variable `TeX-macro-global` must point to your local TeX macro subdirectory (the trailing slash is mandatory).

Close to the bottom of this file you will find the following code:

```
;;; Try to make life easy for MikTeX users.

(when (memq system-type '(windows-nt))
(require 'tex-mik))
```

Comment out the last two lines by adding a semicolon “;” in front of each line. The advantages of `tex-mik.el` or the equivalent `tex-fptex.el` have been included into the patch that you inserted manually. This allows to use the code not only on WinNT, but also on Win95/98/ME.

Save the file and then byte-compile the file by typing **Alt-x byte-compile-file [Return] site-lisp-Path\tex-site.el [Return]**, where site-lisp-Path is the full path of your site-lisp directory, e.g. `C:\Programs\emacsen\site-lisp`. This will create the file `tex-site.elc`.

5. Modify _emacs

   Add the following code to your `_emacs` file to load TeX support at Emacs startup:

```
;; add TeX-support
(load "tex-site")
(custom-set-variables
’(TeX-expand-list (quote (("%p" TeX-printer-query)
("%q" (lambda nil (TeX-printer-query TeX-queue-command 2)))
("%v" TeX-style-check (("^a5$" "windvi %d -paper a5")
("^landscape$" "windvi %d -paper a4r -s 4") ("." "windvi %d")))
("%l" TeX-style-check (("." "latex"))) ("%s" file nil t) ("%t" file t t)
("%n" TeX-current-line) ("%d" file "dvi" t) ("%f" file "ps" t)
("%a" file "pdf" t)))))
```

The first steps with the SGML IDE

... will be postponed to the next chapter because we need some more files until we can really work with the whole system.

Further Reading

An excellent introduction to AucTeX is available as auc-tex.ps.gz (http://gluon.physics.ucla.edu/info/emacs/auc-tex.ps.gz).

Chapter 9. DocBook and HTML document type definitions

The DocBook SGML DTD and Norman Walsh's DocBook DSSSL stylesheets are well suited to write handbooks and other technical documentations (and lots of other stuff too) and publish them either as a set of hyperlinked HTML pages or as a printable document. What you are reading right now has been authored using DocBook.

We will install the latest version of the DocBook SGML DTD. If you intend to work with existing DocBook documents that use older DTDs, you will need to install them as well. The procedure is exactly the same as outlined below.

Norman Walsh's DocBook stylesheets contain a Perl script for the generation of an index. Although it may seem a bit exaggerated to install Perl just to run a single script, there is no way out if you want to automatically create indices for DocBook documents, and probably you will find more useful scripts to feed your Perl interpreter with.

The HTML DTDs are also included in this chapter for two reasons. First, they are useful to write HTML pages that comply with the W3C specifications. Second, most of the necessary files are already on your harddisk as they are included in the SP suite.

Get the files

1. DocBook SGML DTD (http://www.oasis-open.org/docbook/sgml/4.1/docbk41.zip)
2. DocBook DSSSL stylesheets (http://sourceforge.net/projects/docbook/)
3. DocBook DSSSL stylesheet documentation (http://sourceforge.net/projects/docbook/)
4. ISO entity set (http://www.oasis-open.org/cover/ISOEnts.zip)
5. Perl for Win32 (http://www.activestate.com/ActivePerl/download.html)

Note: It is not necessary to download the HTML DTDs. They are already on your disk if you installed OpenJade.

Some general remarks on DTDs and catalogs

For first-time users it is often hard to understand how DTDs, catalog files, OpenJade/onsgmls, and PSGML interact, and to understand what is wrong when they refuse to interact properly. This section briefly describes the general setup of DTDs, before we will install the DocBook DTD and HTML DTDs as examples.

When we installed the SP suite, we created an environment variable called SGML_CATALOG_FILES. This is simply the full path of a catalog file or a list of such full paths. These catalog files map public

identifiers of DTDs to actual files that a SGML-processing application can access. It is mainly a matter of taste whether you use one catalog file or more. Using one catalog file keeps all information in one place, but it requires more manual work if you add or update DTDs.

In the simple example file `test.sgml` that you created initially, we wrote a small SGML document that carried its document type definition at the beginning of the document file. This is fine for small and custom DTDs, but it is inefficient if many documents use the same DTD. Therefore SGML allows to keep the DTDs in separate files and to reference these external files at the beginning of a SGML document. This reference may look like this:

```
<!DOCTYPE HTML SYSTEM "html.dtd">
```

This translates to: The document type of this document is *HTML*, and the DTD which describes this document type is available on the local system in a file called `html.dtd`. It is assumed that `html.dtd` is in the same directory as the document. This basically works fine, but has one major drawback: These documents are not easily portable.

The use of catalog files overcomes this limitation. The corresponding prolog may look like this:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
```

This translates to: The document type of this document is *HTML*, and the DTD has the formal public identifier *-//IETF//DTD HTML 2.0//EN*. This is where catalog files come into play. It is expected that a catalog file (accessible via the SGML_CATALOG_FILES environment variable) has an entry which resolves this identifier to a local file or an URL. The corresponding entry in a catalog file might look like this:

```
PUBLIC "-//IETF//DTD HTML 2.0//EN" "html/html.dtd"
```

This now translates to: The DTD which is referenced with the formal public identifier *-//IETF//DTD HTML 2.0//EN* is stored locally in the file `html.dtd`. The exact path of this file is always relative to the catalog file. In this example the file `html.dtd` can be found in the `html` subdirectory of the directory that contains the catalog file.

Taken together, the idea of installing a DTD on the local system is to:

1. Put the DTD file(s) in some directory
2. Provide a new or edit an existing catalog file which resolves the identifier to a local file
3. Make the catalog file accessible via the SGML_CATALOG_FILES environment variable.

The procedure described so far will enable openade, onsgmls, and PSGML to access DTDs via catalog files. Before a DTD can be used, it has to be parsed to create a representation in the memory. This

parsing can take quite a long time, which is especially a problem with PSGML: the internal parser is implemented in the interpreted language Emacs Lisp. Therefore PSGML provides a mechanism to store a memory representation of a parsed DTD in a separate file and read this instead of the original DTD. Just as the original DTDs, the pre-parsed DTDs can be accessed via catalog files as well. As only PSGML uses this kind of DTD, there is no environment variable to locate these catalog files. This is set in the `_emacs` file instead. The basic steps to make use of this PSGML feature are:

1. Load a document that uses the DTD (or create a new, empty one) and parse the DTD with PSGML
2. Save the parsed DTD with PSGML into the directory that contains the DTD
3. Create or edit a catalog file (usually called `ecatalog` to distinguish it from regular catalog files) which resolves the DTD identifier to the parsed DTD.
4. Modify your `_emacs` to tell PSGML where the `ecatalog` files can be found.

Now, whenever you open or create a SGML document, PSGML first checks whether a parsed version of the given DTD exists and uses it if present. If no parsed version exists, it uses the DTD itself and saves a parsed version for the next time you open a file with this DTD. Whether or not the additional hassle with ecatalog files pays off depends on the speed of your computer and the size of the DTDs that you use.

> **Note:** This automatic saving of the parsed version may fail if you do not have sufficient access rights. In that case, you should log in as an administrator and perform the first two steps in the procedure above for each DTD that you want to access through an ecatalog entry.

We will now go ahead and use this knowledge to install a few useful DTDs.

Install the HTML DTDs

1. Access the DTDs via the menu

   PSGML allows to insert the document type declaration via menu commands. To this end, all DTDs that you want to use this way have to be added to a variable in your `_emacs`. Insert the following lines into your configuration file:

```
;; PSGML menus for creating new documents
(setq sgml-custom-dtd
'(
  ( "HTML 2.0"
    "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2.0//EN\">")
  ( "HTML 2.0 Strict"
    "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2.0 Strict//EN\">")
  ( "HTML 2.0 Level 1"
    "<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML 2.0 Strict Level 1//EN\">")
  ( "HTML 3.2 Final"
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 3.2 Final//EN\">")
  ( "HTML 4"
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\">")
```

```
  ( "HTML 4 Frameset"
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Frameset//EN\">")
  ( "HTML 4 Transitional"
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">")
  ( "DocBook 4.1"
   "<!DOCTYPE Book PUBLIC \"-//OASIS//DTD DocBook V4.1//EN\">")
  )
)
```

Strictly speaking, the last entry does not yet belong here, but we will need it when we install the DocBook DTD. Just keep in mind that we inserted these lines already at this point.

2. Create the directory structure

   Having all DTDs in a separate directory tree simplifies the maintenance greatly. Therefore create a new `\sgml\dtd` subdirectory in a directory which has read access for all users on your system, e.g. `C:\user\default`. This will be the root directory of all DTDs.

3. Copy the files

   Create a subdirectory `html` and copy all relevant files (`html*.*`, `isolat1.*`, and `xml.*`) from the `OpenJade-1.3\pubtext` installation directory into your new `html` subdirectory.

4. Create the catalog file

   Open a new catalog file in the same directory using Emacs, e.g. by typing **C-x C-f c:/user/default/sgml/dtd/html/catalog [RETURN]**. Insert the following lines:

```
OVERRIDE YES

PUBLIC "-//IETF//DTD HTML 2.0//EN" "html.dtd"
PUBLIC "-//IETF//DTD HTML 2.0 Strict//EN" "html-s.dtd"
PUBLIC "-//IETF//DTD HTML 2.0 Level 1//EN" "html-1.dtd"
PUBLIC "-//IETF//DTD HTML 2.0 Strict Level 1//EN" "html-1s.dtd"
PUBLIC "-//W3C//DTD HTML 3.2 Final//EN" "HTML32.dtd"
PUBLIC "-//W3C//DTD HTML 4.0//EN" "HTML4-s.dtd"
PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "HTML4.dtd"
PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN" "HTML4-f.dtd"
PUBLIC "-//W3C//ENTITIES Latin1//EN//HTML" "HTMLlat1.ent"
PUBLIC "-//W3C//ENTITIES Special//EN//HTML" "HTMLspec.ent"
PUBLIC "-//W3C//ENTITIES Symbols//EN//HTML" "HTMLsym.ent"
```

   The first line tells a catalog-using application that the public identifier should override any system identifier. The HTML4 DTDs use URLs as system identifiers for the entity files and thus would need an internet connection whenever you validate them, but with this little trick they will be just as happy with the local copies.

   As the DTD and entity files are in the same directory as the catalog file, a path is not necessary. However, if you want to use a single catalog file for all your DTDs and put it e.g. into the `\sgml\dtd` directory, you would have to specify the filenames as e.g. `/html/html.dtd`. In the following procedures I assume that you use several catalog files to simplify the descriptions.

5. Register the catalog file

   Add the full path of your new catalog file, e.g. `C:\user\default\sgml\dtd\html\catalog`, to the semicolon-separated list of your environment variable SGML_CATALOG_FILES. Use the procedure as described previously.

6. Create your ecatalog file

   Create a new file called `ecatalog` in the same directory as the previous catalog file, e.g. by typing **C-x C-f C:/user/default/sgml/dtd/html/ecatalog [RETURN]** in your Emacs. Enter the following lines into this new file:

   ```
   PUBLIC "-//IETF//DTD HTML 2.0//EN" "html.ced"
   PUBLIC "-//IETF//DTD HTML 2.0 Strict//EN" "html-s.ced"
   PUBLIC "-//IETF//DTD HTML 2.0 Level 1//EN" "html-1.ced"
   PUBLIC "-//IETF//DTD HTML 2.0 Strict Level 1//EN" "html-1s.ced"
   PUBLIC "-//W3C//DTD HTML 3.2 Final//EN" "HTML32.ced"
   PUBLIC "-//W3C//DTD HTML 4.0//EN" "HTML4-s.ced"
   PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "HTML4.ced"
   PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN" "HTML4-f.ced"
   ```

7. Register your ecatalog file

   Put the following lines into your `_emacs`:

   ```
   ;; ecat support
   (setq sgml-ecat-files
     (list
       (expand-file-name "c:/user/default/sgml/dtd/html/ecatalog")
       (expand-file-name "c:/user/default/sgml/dtd/docbook41/ecatalog")
   ))
   ```

   Again, the last line is not related to the HTML DTDs, but we will need it later. Restart your Emacs to let the changes in `_emacs` take effect.

8. Parse the DTDs

   To test-drive the new setup, create a new file in your Emacs with the extension `.html`, e.g. by typing **C-x C-f c:/user/myself/test.html**. This should load PSGML automatically. Now use e.g. the DTD⟶Insert DTD⟶HTML 4 menu command to insert the document type declaration into your test document. PSGML should automatically start to parse the DTD (if not, use the DTD⟶Parse DTD menu command or simply press **C-c C-p**). As PSGML finds a corresponding entry in the registered ecatalog file but no parsed DTD, it will automatically save the DTD as `HTML4s.ced` in the `\html` subdirectory.

   If this test was successful, all other parsed DTDs will be generated on the fly whenever you use a registered DTD for the first time.

   > **Note:** On Windows NT/2000/XP, it may be necessary to create the compiled DTDs with administrator rights (this depends on the access rights of the directories that contain the DTD

files). If this is the case on your system, you should repeat the above procedure for all installed DTDs from your administrator account.

9. Set the SGML declaration

   Open your `_emacs` file and locate the function declaration:

   ```
   (defun sgml-html-mode()
   ```

   Within this function, locate the line:

   ```
   sgml-declaration          "d:/user/default/sgml/dtd/html/html4.dcl"
   ```

   Adjust the path to your local system.

   Note: We just set a reasonable default value in `_emacs`. The declaration you enter here should match the HTML DTD that you will use most of the time. If it should be the case that you mainly use e.g. the HTML3.2 DTD, you would rather specify `html32.dcl` here. Whenever you need a different declaration than the default, use the menu command SGML⟶User options⟶Declaration to set a new value.

Install the DocBook SGML DTD

1. Unzip the files

   Extract the contents of the DocBook DTD archive into a new directory in your SGML tree, e.g. `c:\user\default\sgml\dtd\docbook41`. Using the version number in the subdirectory name simplifies the installation of several versions in parallel (you might need older versions for reasons of compatibility with older, existing documents).

2. Register docbook.cat

   The DTD archive contains a catalog file named `docbook.cat` which must be added to the SGML_CATALOG_FILES environment variable. Use the procedure as described previously. Append the full path (including the filename) of the catalog file, e.g. `c:\user\default\sgml\dtd\docbook41\docbook.cat`, to the semicolon-separated list.

3. Create `ecatalog`

   Use Emacs to create the file `ecatalog` in the DocBook directory, e.g. `c:\user\default\sgml\dtd\docbook41`. Insert the following line:

   ```
   PUBLIC "-//OASIS//DTD DocBook V4.1//EN" "docbook.ced"
   ```

   and save the file.

4. Set the SGML declaration

In your `_emacs`, locate the line:

```
sgml-declaration          "d:/user/default/sgml/dtd/docbook41/docbook.dcl"
```

and adjust the path to your local system.

> **Note:** We just set a reasonable default value in `_emacs`. The declaration you enter here should match the DocBook DTD that you will use most of the time. If it should be the case that you mainly edit vintage DocBook 3.1 documents, you would rather specify `docbook31/docbook.dcl` here. Whenever you need a different declaration than the default, use the menu command SGML⟶User options⟶Declaration to set a new value.

Install the DocBook DSSSL stylesheets

1. Unzip the archive

   Extract the archive containing the DocBook DSSSL stylesheets into a new stylesheets subdirectory, e.g. `c:\user\default\sgml\stylesheets`. This will create a new subdirectory `docbook`. If you already have an older version of the stylesheets in this directory, you may want to rename it to e.g. `db174b` (according to the release) before extracting the new version. This makes it easier to revert to the previous version if you should need to do so.

2. Register catalog

   The DocBook stylesheets archive contains a catalog file named `catalog` which must be added to the SGML_CATALOG_FILES environment variable. Use the procedure as described above.

3. Unzip the documentation

   Unpack the stylesheet documentation archive into a convenient directory, e.g. `C:\user\default\sgml\doc\dbstylesheets`.

Install the ISO entity sets

1. Extract the files

   Extract the contents of the ISO entity archive into a new directory in your SGML tree, e.g. `C:\User\default\SGML\ISOents`. These standardized entities are referenced by DocBook, but as they can be shared by other DTDs, it is better to keep them apart from DocBook proper.

2. Modify docbook.cat

   The file `docbook.cat` that you installed with the DocBook DTD contains the references to the ISO entities. However, you will have to adjust the paths and the filenames. E.g. modify the following line:

```
PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN" "iso-dia.gml"
```

to read:

```
PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN" "../../ISOents/isodia"
```

Look up all ISO entity files which appear in `docbook.cat` and modify the entries accordingly.

Install Perl

1. Run the setup program

   Run the downloaded executable by double-clicking its icon in the folder. This will run the wizard which guides you through the installation. Select the options to associate `.pl` files with the Perl interpreter and to add the Perl directory to your PATH environment variable.

The first steps with the HTML DTDs

Let us now create a new HTML document. Type **C-x C-f ~/htmltest.html** and insert a formal public identifier via the DTD menu. Now use the Markup⟶Insert Element menu commands and related commands to build a HTML document. You can preview the document in your browser while editing. Don't forget to hit the Refresh button of your browser after changes in the source code. Use the SGML⟶Validate menu command to validate your document against the DTD. Make sure that you use the correct SGML declaration, e.g. `HTML4.dcl` if you use one of the HTML4 DTDs.

The first steps with DocBook

Open a new SGML document by typing **C-x C-f ~/dbtest.sgml**. Insert the DocBook formal public identifier via the DTD⟶Insert DTD⟶DocBook 4.1 menu command. Insert some elements into the document, e.g. starting with the chapter and sect1 tags. Enter some para elements to hold some text. Validate your document with the SGMLValidate menu command.

Now prepare the system to create printable output from your document. We'll first do this with Emacs menu commands, and later on the command line. Use the DSSSL⟶File Options⟶Jade backend menu command to select the TeX backend. Use the DSSSL⟶File Options⟶DSSSL stylesheet menu command to set the stylesheet. Use the filename completion feature in the minibuffer to select the file `docbook.dsl` in the `\print` subdirectory of the DocBook hierarchy. Now use the DSSSL⟶Jade command to run Jade with the selected options on your document. Use the DSSSL⟶Recenter output buffer menu command to see the Jade stderr output. Except the "DTDDECL catalog entries are not

supported" message which is a known limitation of Jade there should be no error messages, and Jade should exit with a "Jade finished at" message. If no errors occurred, there will be a new file `dbtest.tex` in your present working directory.

> **Note:** While the abovementioned DTDDECL warning is a nice indicator that OpenJade does *not* hang while processing your 150 kb document, you may find it distracting after a while. To get rid of this message, scan all your catalog files for lines starting with *DTDDECL*. Comment these lines out by adding a " — " (space dash dash space) to the start and the end of the line.

Now use the DSSSL⟶Jadetex menu command to run the intermediate TeX file through LaTeX. This may take some time if new fonts have to be created and is accompanied by a bunch of messages in the output buffer. You should eventually see a successful message which also tells you how many pages were created. Use the DSSSL⟶View DVI command to display `dbtest.dvi`. The curious may load the intermediate `dbtest.tex` file into Emacs to see the fancy output of the TeX backend.

Now hit DSSSL⟶dviPS to create a PostScript document. Click on DSSSL⟶View PS to view the PostScript document `dbtest.ps` with GhostView.

Modify the file options to create HTML output from the same DocBook source. Using the commands described above, set the OpenJade backend to SGML and set the stylesheet to `docbook.dsl` in the `\html` subdirectory of your DocBook stylesheet hierarchy. Now run Jade again. This will create a set of HTML files with a file like `book1.htm` as the starting point.

To process your document on the command line you essentially use the same commands as in the OpenJade chapter. The main difference is that we need to supply a proper SGML declaration as the newer DocBook versions do not work with OpenJade's builtin default declaration. The correct one to use is `docbook.dcl` which is part of the DocBook DTD archive. To transform your DocBook document (aptly named `docbook.sgml` in these examples) into either RTF or HTML, run the following commands:

```
c:\user\myself> openjade -t rtf -d \path\to\print\docbook.dsl \path\to\docbook.dcl doc-
book.sgml
```

```
c:\user\myself> openjade -t sgml -d \path\to\html\docbook.dsl \path\to\docbook.dcl doc-
book.sgml
```

Further Reading

The DocBook DTD documentation (http://www.oasis-open.org/docbook/documentation/index.html) is available at OASIS.

Norman Walsh has written *DocBook: The Definitive Guide*, which is available as a dead-tree version (http://www.oreilly.com/catalog/docbook/) and online (http://www.oreilly.com/catalog/docbook/chapter/book/docbook.html).

Dave Pawson maintains a DocBook FAQ (www.dpawson.co.uk/docbook/).

If you want to customize the DocBook stylesheets, you don't want to miss the DocBook stylesheet documentation (http://sourceforge.net/projects/docbook). These documents also contain a very clear and concise description of the index creation process with the `collateindex.pl` script, see the file

`indexing.htm`. Another good source about stylesheet-related questions is here (http://www.miwie.org/docbook-dsssl-faq.html).

The Text Encoding Initiative (http://www.tei-c.org) has published a DTD which seems more geared towards linguistic applications. There is also a somewhat stripped-down version called TEI Lite (http://www.uic.edu/orgs/tei/lite). Richard Light (with some additions by Jon Bosak) has published TEI Lite DSSSL print stylesheets (ftp://sunsite.bcc.bilkent.edu.tr/pub/sun-info/standards/dsssl/stylesheets/tei/tei-dsl.zip), but they don't seem to be actively maintained (there are actively maintained XSL stylesheets for the XML version of the DTD, though).

DSSSL stylesheets for HTML come in handy if you are not satisfied with what your browser prints. Stylesheets are available for HTML 3.2 (ftp://sunsite.bcc.bilkent.edu.tr/pub/sun-info/standards/dsssl/stylesheets/html3_2/html32hc.zip).

IV. XML processing

XML has attracted a far greater number of programmers than SGML, and the result is a nice pool of applications to choose from (for an overview, see e.g. Lars Marius Garshol's page about Free XML tools and software (http://www.garshol.priv.no/download/xmltools/)). One of the reasons is certainly that XML was specifically designed to be easier to parse than SGML. The downside is that putting together an XML processing system needs a few decisions upfront. The following chapters will give you a (necessarily subjective) choice of several XML parsers and XSLT processors with various output types. You may choose what you need, or install all of them and compare. The table below is intended to give some guidance.

> **Note:** You should also be aware that most processors implement some sort of extensions to the standards. Some XSLT stylesheets require specific extensions for some special tricks, e.g. for chunking HTML output. Consult the documentation of the stylesheets you plan to use to find out which processor you should prefer.

The main considerations when picking one of the combos are:

- Parser interface: There are two accepted standards for XML parser interfaces: SAX and DOM. The difference between these two models in simple words is as follows: a SAX-capable parser calls a registered function for each start tag, end tag, and the data inbetween. The parsing is done sequentially, so there is no need to have the whole document in memory at any time (in some cases it is not even necessary to have the whole document available, it may be received in chunks). The downside is that the elements have to be processed as they are encountered during parsing. If an application needs access to previous or later elements, it has to do some sort of buffering. On the other hand, a DOM-capable parser creates an in-memory representation of the whole document. This may need a lot of memory for large documents, but all elements can be accessed freely at any time during processing. For you as the end-user the parser interface issue boils down to the question which XSLT engine can be used with which parser.
- Validating vs. non-validating: In the XML world documents do not have to be valid (in contrast to SGML), but they can be validated against a DTD if necessary. Therefore you can use either validating and non-validating parsers depending on your needs.
- XML uses Unicode to encode characters. The programming language Java also builds on Unicode from the ground up, so processing XML with Java is kind of a natural match. The only downside is that Java programs tend to be a little slower than C/C++ programs and that you need the Java Runtime Engine (a bytecode interpreter) to run the applications. C/C++ programs are faster and have a smaller footprint, but programming Unicode in C or C++ is just not as popular.

Table 1. XML parsers and XSLT processors covered in this tutorial

Name	Parser interface	Validating parser	Language
xsltproc	DOM, SAX	yes	C
XP/XT	SAX	no	Java
Xerces/Xalan	DOM, SAX	yes	Java
Saxon/Ælfred	SAX	?	Java

None of the mentioned XSLT processors can directly create printable output (all do HTML output, though). Therefore we need a set of additional applications to transform our XML documents into PDF and RTF files.

> **Note:** XML differs from SGML in that a SYSTEM identifier for the DTD file is mandatory. In order to keep the files portable, usually a URL is specified for this purpose instead of a local path. This means that for most XML transformations an internet connection is mandatory. It is not necessary for editing XML files with PSGML as PSGML does not attempt to resolve URLs.

Chapter 10. xslide

Just as we previously taught Emacs to nicely edit SGML and XML files with the PSGML package, we will now install a package that enhances our user experience with XSLT stylesheets.

Get the files

1. Xslide (http://www.menteith.com/xslide/)

Install xslide

1. Extract the xslide archive

   Extract the xslide archive into a temporary directory like `c:\temp`.

2. Copy Emacs lisp files to the Emacs `site-lisp` directory

   Copy all `.el` files and the file `xslide-initial.xsl` to your common Emacs `site-lisp` directory, e.g. `C:\Programs\emacsen\site-lisp`

3. Byte-compile the Emacs lisp files

   In Emacs, run the following commmand **Alt-x byte-compile-file *`path/to/xslide-file`*** in turn for each of the files `xslide.el`, `xslide-abbrev.el`, `xslide-data.el`, `xslide-font.el`, `xslide-process.el`.

4. Modify `_emacs`

   Insert the following code into your `_emacs` file:

```
;; XSL mode (using the xslide package)
(autoload 'xsl-mode "xslide" "Major mode for XSL stylesheets." t)

;; Turn on font lock when in XSL mode
(add-hook 'xsl-mode-hook
   'turn-on-font-lock)

(setq auto-mode-alist
      (append
       (list
 '("\\.fo" . xsl-mode)
 '("\\.xsl" . xsl-mode))
       auto-mode-alist))

;; Uncomment if using abbreviations
;; (abbrev-mode t)
```

The first steps with PSGML and XSlide

Just as we did in the case of PSGML we should prepare a simple XML document with an accompanying stylesheet to get used to the editing environment and to have something to test the shortly-to-be-installed parsers and processors with. We will use essentially the same document as in the PSGML example, adjusted to the XML syntax. This example also does not use an external DTD to simplify processing. There is no need to understand the syntax of the DTD at the moment, simply type it or copy it to your document.

Write an XML document with PSGML

Restart Emacs to apply the latest changes to your `_emacs` and open a new XML document: Type **C-x C-f ~/test.xml** to create the document `test.xml` in your home directory (you may as well type the full path of a file in any other convenient directory). Emacs should load the XML mode (check the modeline close to the bottom of the Emacs window). Now type or paste the following DTD into the new buffer:

```
<?xml version="1.0"?>
<!DOCTYPE HTMLLite [
<!ELEMENT HTMLLite (H1|P)* >
<!ELEMENT H1 (#PCDATA|EM|STRONG)* >
<!ELEMENT P (#PCDATA|EM|STRONG)* >
<!ELEMENT EM (#PCDATA) >
<!ELEMENT STRONG (#PCDATA) >
]>
```

As you just have created the DTD, PSGML is not yet aware of it (when you load an existing XML file, PSGML will automatically parse the DTD due to a setting in our `_emacs` file). Press **C-c C-p** to parse the DTD now. Check your DTD again if PSGML issues an error.

Now place the cursor at the end of the buffer and start to append the contents of the document to the prolog. Use the menu command Markup⟶Insert Element to enter the first element. You will notice that PSGML offers only one possible element to insert: HTMLLite. Accept this and insert another element right where you are. This time you will see that you can either insert a H1 or a P element. You can enter some text between the H1 or P start and end tags. Try to insert another element when the cursor is inside the text in a H1 or P element. This will allow you to insert either a STRONG or an EM element, which again can hold some text. Try also the keyboard shortcut for inserting an element, **C-c C-e**. You will be prompted for the name of an element. Either enter the full name of an element, the first letters and then the tab key, or simply press the tab key twice to see a list of possible tags at the given location. Note that you don't have to enter hard returns or to manually indent, PSGML takes care of this.

When you're done, save your document by pressing **C-c C-s**. A valid HTMLLite document may look like this:

```
<HTMLLite>
  <H1>Character of Constantine</H1>
  <P>The person, as well as the mind, of Constantine had been enriched by
     nature with her choicest endowments.</P>
```

```
  <P>His stature was <EM>lofty</EM>, his countenance <EM>majestic</EM>, his
     deportment <EM>graceful</EM>; his strength and activity were displayed
     in every manly exercise, and, from his earliest youth to a very
     advanced season of life, he preserved the vigour of his constitution
     by a <STRONG>strict</STRONG> adherence to the domestic virtues of
     chastity and temperance.</P>
</HTMLLite>
```

You should also have noticed that PSGML performs syntax-highlighting: Start/end tags and other language-related constructs are shown in different colors to increase readability. Do not delete your test document yet, because you may wish to reuse it in the next chapters.

Writing an XSLT stylesheet with xslide

Now let us create a matching stylesheet for our XML document. We cannot do much with it right now besides creating it, but you'll have something to play with in the next chapters.

Create a new `test.xsl` file in the same directory that contains your `test.xml` document: Type **C-x C-f ~/test.xsl** (or whatever path you chose previously). You should be pleasantly surprised to see that xslide automatically inserts a skeleton stylesheet for HTML output into a new, empty buffer.

The skeleton stylesheet processes the root element ("/") and creates the `HTML` and `BODY` elements of the output file. If you wish, you can enhance the output by adding a `HEAD` element with a `TITLE`.

Move your cursor just after the `xslt:template` end tag and press **C-c-<**. This will invoke the **xsl-insert-tag** command. You can use the tab-completion in the minibuffer to create another `xsl:template` element. As an attribute, enter `match="H1"`. Without much surprise, this template will process the HTMLLite `H1` elements. As the content of this template, enter:

```
    <h1>
      <xsl:apply-templates/>
    </h1>
```

This will simply map the HTMLLite `H1` element to the HTML `H1` element.

Now add the other missing templates in the same way. You should end up with a stylesheet that looks somewhat like this:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:output method="html"/>

  <xsl:template match="/">
    <html>
      <head>
```

```
        <title>HTMLlite document</title>
      </head>
      <body bgcolor="#FFFFFF">
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="H1">
    <h1>
      <xsl:apply-templates/>
    </h1>
  </xsl:template>
  <xsl:template match="P">
    <p>
      <xsl:apply-templates/>
    </p>
  </xsl:template>
  <xsl:template match="EM">
    <i>
      <xsl:apply-templates/>
    </i>
  </xsl:template>
  <xsl:template match="STRONG">
    <b>
      <xsl:apply-templates/>
    </b>
  </xsl:template>
</xsl:stylesheet>
```

Further Reading

Due to the popularity of XML and XSLT, there is not exactly a lack of books and online resources about these topics. One example of a fairly concise and pleasant introduction into XSL and XSLT can be found here (http://www.brics.dk/~amoeller/XML/transformation/index.html).

Summary

Now you should be able to:

- Write simple XML documents
- Write simple XSLT stylesheets

Chapter 11. xsltproc: a XSLT engine in C

xsltproc is a small piece of a larger effort: it is just one application of the libxsl package which in turn uses the libxml package to parse xml. Both are C libraries intended to build XML/XSL-capable applications, but we'll use only xsltproc as the probably smallest available parser/processor combo. As xsltproc is implemented in C, there is no need to have the Java Runtime Engine installed, and it should run faster especially on old hardware.

Get the files

1. libxml/libxslt Win32 binaries (http://www.fh-frankfurt.de/~igor/projects/libxml/index.html). You need the libxml, the libxslt, and the iconv binaries packages.

Install xsltproc

1. Install the binary and the libraries

   Create a new subdirectory in a convenient location, e.g. `C:\Programs\xsltproc`. Copy the following executables and libraries from their archive files to the new `xsltproc` directory:

`libxslt-1.0.3-win32.zip`

: `xsltproc.exe`, `libexslt.dll`, and `libxslt.dll`

`libxml2-2.4.3.win32.zip`

: `libxml2.dll` (the smaller, non-debug version in the archive)

`iconv-1.7.win32.zip`

: `iconv.dll`

2. Adjust PATH

   Add the new `C:\Programs\xsltproc` to your PATH environment variable. Use the procedure outlined previously.

The first steps with xsltproc

To see what xsltproc can do for us, we'll use the XML document and stylesheet that we created in the previous chapter. We will try to generate a HTML rendering of the XML document.

To this end, change to the directory that contains these files and run the following command in your shell:

```
C:\user\myself>xsltproc test.xsl test.xml > test.html
```

View the resulting `test.html` file in your favourite browser. You should see a nicely formatted HTML representation of your XML document.

Further reading

The usage of xsltproc is explained in some detail here (http://xmlsoft.org/XSLT/xsltproc.html).

Chapter 12. Saxon, XT, Xalan: Java-based XSLT engines

This chapter presents three Java-based XSLT engines with their XML parsers. You can install all of them, or choose the one most suitable for your needs.

XP and XT were written by one of the most renowned programmers in the SGML/XML field, James Clark. The parser is non-validating which helps it to be the fastest XML parser implemented in Java (so the author claims).

Xerces and Xalan are being developed under the same virtual roof as the Apache web server and other web-related tools. Xerces is a full-fledged validating XML parser with DOM and SAX interfaces. Xalan is a matching XSLT engine. Implementations in both C++ and Java are available. We'll use the Java versions of both tools here.

Saxon is actually both a Java library for XML processing and a XSLT engine. We will look at the latter functionality only, though. Saxon ships with the Ælfred XML parser.

To run these Java applications, we will install the latest version of Sun's Java Runtime Engine. Before you go ahead you should check whether your system already has this JRE installed, as browsers like Netscape and Mozilla use this to run Java applets. Browse the software list in the "Add/remove software" applet in the system control panel to see which version of the JRE, if at all, is installed. If you have a real old version (something like 1.1) you'll still have to update.

Get the files

1. XP (ftp://ftp.jclark.com/pub/xml/xp.zip)
2. XT (ftp://ftp.jclark.com/pub/xml/xt.zip)
3. Xalan (Java) (http://xml.apache.org/dist/xalan-j/). This archive contains all necessary classes including the Xerces parser.
4. Saxon 6.4.4 (http://saxon.sourceforge.net) Choose the full version, not the smaller, but slower, instant version.
5. Java Runtime Engine (http://java.sun.com/j2se/1.3/jre/download-windows.html)

Install the Java Runtime Engine

1. Run the installer

   Run the self-extracting binary that you downloaded. This will use the familiar InstallShield wizard to ask you a few questions. Select a suitable installation directory, e.g. `C:\Programs\Javasoft`, and you're basically done.

2. Adjust PATH

In order to run the Java command-line applications conveniently, add the `bin` directory, e.g. `C:\Programs\Javasoft\JRE\1.3.1\bin` to your PATH environment variable. Use the procedure outlined previously.

3. Fix CLASSPATH

   In older versions of the Java Runtime Engine, the environment variable CLASSPATH was used to locate additional Java classes. The use of this environment variable is discouraged in newer versions, and a command-line switch for `java` and related tools should be used instead. To avoid conflicts, you should check whether this variable is set on your system. Unless specific tools other than the Java tools need it, you should remove it.

Install the XP and XT Java classes

1. Create a class repository

   To simplify maintenance, all your Java classes should be installed in a single directory tree. Unless you already have such a repository, you should create a directory right now, like `C:\Programs\java`.

2. Extract the archives

   Extract the file `xp.jar` from the XP archive and copy it to your Java class repository, e.g. `C:\Programs\java`. Extract the files `xt.jar` and `sax.jar` from the XT archive and copy them to the same location.

Install the Xerces and Xalan Java classes

1. Extract the archives

   Extract the files `xerces.jar` and `xalan.jar` from the Xalan archive and copy them to your Java class repository, e.g. `C:\Programs\java`.

Install the Saxon and Ælfred Java classes

1. Extract the archives

   Extract the files `saxon.jar`, `saxon-fop.jar`, and `saxon-jdom.jar` from the Saxon archive and copy them to your Java class repository, e.g. `C:\Programs\java`.

The first steps with the Java-based tools

The first test runs with these tools will also not be too exciting, but rest assured that it will work all the same if we later use "real" documents like DocBook XML documents.

Change to the directory that contains the test files that we created previously and run the following commands in your shell (the backslashes at the end of the lines denote continuation of the line - don't type them in):

```
C:\user\myself>java -cp "C:\Programs\java\xt.jar;C:\Programs\java\xp.jar" \
com.jclark.xsl.sax.Driver test.xml test.xsl > test.html
```

```
C:\user\myself>java -cp "C:\Programs\java\xalan.jar;C:\Programs\java\xerces.jar" \
org.apache.xalan.xslt.Process -in test.xml -xsl test.xsl -out test.html
```

```
C:\user\myself>java -cp "C:\Programs\java\saxon.jar;C:\Programs\java\saxon-fop.jar; \
C:\Programs\java\saxon-jdom.jar" com.icl.saxon.StyleSheet test.xml test.xsl \
-o test.html
```

View the resulting `test.html` file in your favourite browser. You should see a nicely formatted HTML representation of your XML document.

Further Reading

All packages ship with extensive documentation in HTML format.

Chapter 13. Creating printable output

While Jade/OpenJade use built-in backends to create HTML and printable output, XSLT engines use a different path to arrive at the same end: They are capable to transform the XML document to “Formatting Objects”. The result is a temporary XML document which contains all data and the necessary formatting informations. This temporary document is fed to another application (a formatting objects processor) to create the final printable output.

FOP is a Java application to directly create PDF, MIF, PCL, or plain text files from FO output. Alternatively, if you pass a XML document and a XSL stylesheet, FOP will use Xerces and Xalan to do the XSLT transformation on the fly.

JFOR is another Java application which creates RTF files from FO output. RTF files can be viewed and printed with most word processors.

xmltex and the PassiveTeX macros play a similar role for XML as Jade’s TeX backend and the JadeTeX macros play for SGML (the macros are actually closely related). PassiveTeX uses the FO output of an XSLT engine to generate PDF output. While this sounds more complicated than directly creating PDF, this approach makes use of the superior layout capabilities of the TeX system. The macros were already installed with the TeX installation, so we just need to look at a few examples how to use them.

Get the files

1. FOP (http://xml.apache.org/dist/fop/)
2. JFOR (http://sourceforge.net/projects/jfor). Pick the latest binary package (`jfor-x.y.z.jar`)

Install FOP

1. Extract the archive

   Extract the files `fop.jar`, `config.xml`, `userconfig.xml`, `avalon-framework-4.0.jar`, `batik.jar`, `jimi-1.0.jar`, and `logkit-1.0b4.jar` to your Java class repository, e.g. `C:\Programs\java`.

Install JFOR

1. Install the class file

   Copy the JFOR class archive `jfor-0.5.1` to your Java class repository, e.g. `C:\Programs\java`.

The first steps towards printable output

...will have to wait until we have a suitable DTD with stylesheets to play with. The next chapter will provide example transformations of DocBook XML documents.

Further Reading

The Java packages all ship with extensive HTML documentation.

Chapter 14. DocBook XML DTD and XSLT stylesheets

The DocBook XML DTD is the XML counterpart of the DocBook SGML DTD and tries to match the latter as close as the differences between SGML and XML permit. With a few caveats, it is generally possible to migrate any DocBook SGML document to XML and vice versa.

Although you could also use DSSSL stylesheets and a DSSSL engine to transform your XML documents, XSLT has superseded DSSSL in the realm of XML. Fortunately Norm Walsh provides also an XSLT equivalent of his modular DSSSL stylesheets, so at least for the HTML output there is not much difference in the rendering.

Get the files

1. DocBook XML DTD (http://www.oasis-open.org/docbook/xml/4.1.2/docbkx412.zip)
2. docbook-xsl (http://sourceforge.net/projects/docbook/) The XSLT stylesheets

Note: In contrast to the DSSSL stylesheets, the XSLT stylesheets contain their documentation in the same archive, so you don't have to download them separately.

Install the DocBook XML DTD

Note: The procedure to install the XML version of the DTD is pretty similar to the installation of the SGML version. If you set up your system for both SGML and XML processing, chances are that pieces of the `_emacs` code described below already exist. Instead of just pasting the following chunks at the end of `_emacs`, you should modify your existing code by merging the missing pieces.

1. Unzip the files

   Extract the contents of the DocBook XML DTD archive into a new directory in your XML tree, e.g. `c:\user\default\xml\dtd\docbook412`. Using the version number in the subdirectory name simplifies the installation of several versions in parallel (you might need to install or keep older versions for reasons of compatibility with older, existing documents).

2. Register docbook.cat

   The DTD archive contains a catalog file named `docbook.cat` which must be added to the SGML_CATALOG_FILES environment variable. Use the procedure as described previously. Append the full path (including the filename) of the catalog file, e.g. `c:\user\default\xml\dtd\docbook412\docbook.cat`, to the semicolon-separated list.

Locate the line

```
-- OVERRIDE YES --
```

in the catalog file and remove the leading and trailing dashes to uncomment this entry. This will allow PSGML to work smoothly with the XML DTD.

3. Create `ecatalog`

   Use Emacs to create the file `ecatalog` in the DocBook XML directory, e.g. `c:\user\default\xml\dtd\docbook412`. Insert the following line:

   ```
   PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN" "docbookx.ced"
   ```

   and save the file.

4. Register your ecatalog file

   Put the following lines into your `_emacs`:

   ```
   ;; ecat support
   (setq sgml-ecat-files
     (list
       (expand-file-name "c:/user/default/xml/dtd/docbook412/ecatalog")
   ))
   ```

5. Access the DTDs via the menu

   PSGML allows to insert the document type declaration via menu commands. To this end, all DTDs that you want to use this way have to be added to a variable in your `_emacs`. Insert the following lines into your configuration file:

   ```
   ;; PSGML menus for creating new documents
   (setq sgml-custom-dtd
   '(
     ( "DocBook XML 4.1.2"
      "<!DOCTYPE book PUBLIC \"-//OASIS//DTD DocBook XML V4.1.2//EN\" \"http://www.oasis-
   open.org/docbook/xml/4.1.2/docbookx.dtd\">")
     )
   )
   ```

Install the DocBook XSLT stylesheets

1. Unzip the archive

   Extract the archive containing the DocBook stylesheets into a suitable stylesheets subdirectory, e.g. `c:\user\default\xml\stylesheets`. This will create a new subdirectory `docbook-xsl-1.45`. If you want to, you can change the name of the directory to `docbook-xsl`.

This makes updating easier, as you can always rename back your current version and install the newest version as `docbook-xsl`. This way batch files with hardcoded paths will always use the latest installed version , but you still have your “last known good” version just in case.

The first steps with DocBook

Create a document

Open a new XML document by typing **C-x C-f ~/dbtest.xml**. Insert the DocBook document type declaration via the DTD⟶Insert DTD⟶DocBook XML 4.1.2 menu command. Insert some elements into the document, e.g. starting with the chapter and sect1 tags. Enter some para elements to hold some text. Validate your document with the SGMLValidate menu command. Make sure you use the SGML declaration for XML documents (`xml.dcl`).

Create HTML output

HTML output is the simpler transformation as it requires only one application (the XSLT engine). Here is how you can generate a HTML file from your test document with the various XSLT engines (the backslashes at the end of the lines denote continuation of the line - don’t type them in):

```
C:\user\myself>xsltproc C:\user\default\xml\stylesheets\docbook-xsl\html\docbook.xsl \
dbtest.xml > dbtestxp.html

C:\user\myself>java -cp "C:\Programs\java\xt.jar;C:\Programs\java\xp.jar" \
com.jclark.xsl.sax.Driver dbtest.xml \
C:\user\default\xml\stylesheets\docbook-xsl\html\docbook.xsl > dbtestxt.html

C:\user\myself>java -cp "C:\Programs\java\xalan.jar;C:\Programs\java\xerces.jar" \
org.apache.xalan.xslt.Process -in test.xml -xsl \
C:\user\default\xml\stylesheets\docbook-xsl\html\docbook.xsl \
-out dbtestxc.html

C:\user\myself>java -cp "C:\Programs\java\saxon.jar" \
com.icl.saxon.StyleSheet  -o dbtestsx.html dbtest.xml \
C:\user\default\xml\stylesheets\docbook-xsl\html\docbook.xsl
```

Create printable output

Creating printable output from an XML document is a two-step procedure in most cases: First we have to create an intermediate XML file containing formatting objects (the FO file). This is done with an XSLT processor and a suitable stylesheet. The second step needs some sort of formatting objects processor to actually create the printable output from the intermediate FO file. Let’s first see how we can create this FO file with our XSLT engines (you should of course only run those engines that you actually installed).

In case you want to try all of them it is prudent to use different names for the output files each time, as shown here (the backslashes at the end of the lines denote continuation of the line - don't type them in):

```
C:\user\myself>xsltproc C:\user\default\xml\stylesheets\docbook-xsl\fo\docbook.xsl \
dbtest.xml > dbtestxp.fo

C:\user\myself>java -cp "C:\Programs\java\xt.jar;C:\Programs\java\xp.jar"\
com.jclark.xsl.sax.Driver dbtest.xml \
C:\user\default\xml\stylesheets\docbook-xsl\fo\docbook.xsl > dbtestxt.fo

C:\user\myself>java -cp "C:\Programs\java\xalan.jar;C:\Programs\java\xerces.jar" \
org.apache.xalan.xslt.Process \
-in test.xml -xsl C:\user\default\xml\stylesheets\docbook-xsl\fo\docbook.xsl \
-out dbtestxc.fo

C:\user\myself>java -cp "C:\Programs\java\saxon.jar" \
com.icl.saxon.StyleSheet  -o dbtestsx.fo dbtest.xml \
C:\user\default\xml\stylesheets\docbook-xsl\fo\docbook.xsl
```

Now we try to turn our FO file into a PDF file using the PassiveTeX macros. Run the following command, substituting the FO filename as appropriate:

```
C:\user\myself>pdfxmltex dbtest.fo
```

Alternatively we can use FOP to transform the XML document to PDF with a single command (this is the exception to the two-step rule, but you can guess from the classpath that FOP uses Xalan internally):

```
C:\user\myself>java -cp "C:\Programs\java\fop.jar; \
C:\Programs\java\batik.jar;C:\Programs\java\jimi-1.0.jar; \
C:\Programs\java\xalan.jar; C:\Programs\java\xerces.jar; \
C:\Programs\java\logkit-1.0b4.jar;C:\Programs\java\avalon-framework-4.0.jar" \
org.apache.fop.apps.Fop -xsl \
"C:\user\default\xml\stylesheets\docbook-xsl-1.45\fo\docbook.xsl" \
-xml test.xml -pdf test.pdf
```

Finally we can try and see what the RTF output from the FO file looks like (substitute the FO filename as appropriate):

```
C:\user\myself>java -cp "C:\Programs\java\jfor-0.5.1.jar; \
C:\Programs\java\xerces.jar" ch.codeconsult.jfor.main.CmdLineConverter \
dbtest.fo dbtest.rtf
```

Further Reading

The DocBook DTD documentation (http://www.oasis-open.org/docbook/documentation/index.html) is available at OASIS.

Norman Walsh has written *DocBook: The Definitive Guide*, which is available as a book at O'Reilly (http://www.ora.com) and online (http://www.nwalsh.com/docbook/defguide/index.html). While this

book mainly covers the SGML version, it contains all necessary information for the XML version and helpful hints to migrate SGML documents to XML.

V. Concluding remarks

Now that you have gone through a few hours of unzipping files, modifying your system, and editing files with a strange syntax, you should have a running system to write and publish SGML documents. This concluding chapter is intended to clean up the installation and to tell you where to go if something doesn’t work as expected.

Chapter 15. Cleaning up

After completing the installation, there is a final step to do if you're not the sole SGML user on your computer. If the SGML system is to be used by several people on your NT box, it is necessary to move the relevant parts of your _emacs to one of the system-wide customization files `site-start.el` or `default.el`. As pointed out previously, `site-start.el` usually contains the startup code that may be overridden by the users, whereas `default.el` contains the code which should not be overridden.

Chapter 16. What if...

As things like SGML, DSSSL, and probably Emacs and Emacs Lisp are not as trivial as (but more rewarding than) typing something into MS Word, there will most likely remain a lot of questions that this short tutorial couldn't answer. This section covers some of the internet resources which may be helpful in the case of problems.

- General questions about SGML and XML and the usage of related software can be discussed in the comp.text.sgml (news://comp.text.sgml) and comp.text.xml (news://comp.text.xml) newsgroups, respectively.
- Questions about DSSSL and OpenJade will most likely be answered in the DSSSL mailing list (http://www.mulberrytech.com/dsssl/dssslist/). This link will also guide you to the list archive.
- If you look for further information about XSL, have a look at Dave Pawson's XSL FAQ (http://www.dpawson.co.uk/) or join the XSL-List (http://www.mulberrytech.com/xsl/xsl-list/index.html). This link will also guide you to the list archive.
- General questions about TeX, LaTeX, and JadeTeX can be posted to the comp.text.tex (news://comp.text.tex) newsgroup.
- Questions specific to the fpTeX distribution can be discussed in the fpTeX mailing list. To subscribe, visit the fpTeX mailing list info page (http://www.tug.org/mailman/listinfo/fptex). The mailing list is archived (http://www.tug.org/pipermail/fptex/). Send contributions to fptex@tug.org (mailto:fptex@tug.org).
- General questions to Emacs are welcome at the comp.emacs (news://comp.emacs) newsgroup.
- Specific questions about the NTEmacs port of GNU Emacs should be sent to the help-emacs-windows mailing list. To obtain information about this list or to subscribe, please visit the help-emacs-windows info page (http://mail.gnu.org/mailman/listinfo/help-emacs-windows). The mailing list is archived (http://mail.gnu.org/pipermail/help-emacs-windows/). Send contributions to this list to help-emacs-windows@gnu.org (mailto:help-emacs-windows@gnu.org).
- If you have any problems with the manual itself, be it you cannot figure out what the author meant or be it that things appear to be different on your computer, do not hesitate to contact the author (hoenicka_markus@compuserve.com (mailto:hoenicka_markus@compuserve.com)). I will try to incorporate all suggestions, corrections, bugfixes and the like into future versions for the benefit of all readers.

Appendix A. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

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

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

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

1. APPLICABILITY AND DEFINITIONS

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

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

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

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

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

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

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

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

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download

anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

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

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 less than five).

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

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. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

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 as "Endorsements" or to conflict in title with any Invariant Section.

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

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

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

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

5. COMBINING DOCUMENTS

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

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

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

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

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

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

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.

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 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

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