Developing an Open Source Project from Scratch

Jan Schaumann <jschauma@netmeister.org>

September 4, 2001

Abstract

This column is in itself an ongoing project with the purpose of documenting the development of an Open Source Project from scratch. The project monitored and explained in here will be the “MakeMan” (or “mama”) project: “MakeMan” will provide a basic XML interface and a (possibly several different) frontends to generate man pages.

Contents

Listings

1 Introduction

This column is in itself an ongoing project with the purpose of documenting the development of an Open Source Project from scratch. The project monitored and explained in here will be the "MakeMan" (or "mama") project:

"MakeMan" is a project to provide several frontends, GUI and non-GUI, to an XML interface to write man pages. "MakeMan" will use "DocBook"'s *RefEntry* definition as a basis[1], so as to allow a large variety of applications to access the same data. This will make it possible to write different frontends, possibly networked client and www-interfaces, SOAP-services etc. while at the same time promoting proper documentation.

While following the development of this project, the reader will gain a thorough understanding of the various Open Source tools, resources and possibilities used and appreciated all over the world. Since the project is Open Source, the reader can participate in and influence the development of "MakeMan"; (s)he can practice, learn and teach Open Source techniques and technologies.

1.1 Things we will cover

As this documentation and the project develops, we will encounter various tools, concepts, languages; we will learn to use them wisely and properly (or so we hope) and as this entire project is work in progress, nothing is written in stone. Let's try to come up with a list of technologies that we will, might or want to cover.

Topics that (almost) definitely will be covered:

- *SGML[22] / XML[1]* – XML has been The Next Big Thing(tm) for quite a while now, it's about time that we see if it lives up to the expectations. If all goes well, an XML basis will allow us to deploy the same algorithms in different approaches. Naturally, we will need to take a close look at XML's parent, SGML.
- *VIM[2]* – enough of these IDE's already! I'll use my favorite editor for all editing and we shall see if we can't find a few nifty tricks that will make our life easier. If you're an emacs-person, feel free to contribute with *The Emacs Way* every time something turns out to be vim-specific.
- *autoconf[3] and friends* – we want our project to be as platform independent as possible. It is vital to understand how to check the system

[1]initially, we intended to define an XML DTD

for dependencies, how to actually generate the infamous “configure” script,

- *make[4]* – naturally, “make” will become our close friend. Hopefully, this project will develop and grow so that we can learn about complex Makefiles spanning various directories etc.
- *CVS[5]* – to grant access to the code to other Open Source developers, “CVS” will be used.
- *man[6]* – well, obviously, since the end-result will be a tool to develop man pages, we will need to closely understand “man” itself.

A frontend for “MakeMan” will be implemented in at least one of the following:

- *GTK+/GNOME[8][7]* – GUI development in C
- *Qt/KDE[9][10]* – GUI development in C++
- *Java/SWING[11]* – GUI development in Java
- *Perl[12]* – text based tools
- *(n)curses[13]* – console UI development in C
- *Python[14]* – I never did anything in Python, now might be as good a time as ever to start tinkering with it

Additional topics that might be covered:

- *flex[15]* and *bison[16]* – it might be interesting (and possibly needed) to write our own parser to validate a given input-file.
- *PHP[17]* – a web-frontend
- *SOAP[18]* – this might come in handy with a PHP or Perl based web frontend. Yet another buzzword that demands to be investigated.

2 SourceForge

As this project is intended to grow and the feedback and input of other Open Source developers is desired, it might be a good idea to host the project – that is all files, releases, documentation etc. – in a public space as well. Usually it will be fully sufficient to simply maintain a project on your own webserver, however, since *SourceForge[19]* has become quite an elementary part of the Open Source Movement, we should not skip the opportunity to introduce the reader to the possibilities of this great Open Source Repository.

Once you read through the website, the advantages become obvious: you will get access to a cvs-account, web-space, mailinglists etc. You can add and manage other developers to your project and in general develop your software in a much more organized and professional manner than were you to take care of all these administrative tasks yourself.

In order to host a project at SourceForge, one needs a user-account and must fill out an application providing the relevant information for the project. All this is very easy and is explained in great detail on http://www.sourceforge.net.

Once your application has been approved, you will be granted access to a shell-server, from where you can maintain your projects website. On August 14th, 2001, I uploaded the first files to our SourceForge account: the project became alive!

Every project will have a "Unix-name". This is a short name which will be used as a hostname for the website and also specifies your projects entry in the SourceForge database. In this case, the "Unix-name" is "mama", so that you can access the official project website at http://mama.sourceforge.net while the administrative information for the project is available at http://www.sourceforge.net/projects/mama/

3 The Basis: SGML

As stated earlier, the basis of this project will be XML. After having drafted a first hierarchical structure of the document[2], Open Source started to happen: Somebody emailed me informing me that there already exists a DTD for man pages. The *DocBook[20]* project includes a definition for a "RefEntry", which "is an appropriate wrapper for any small unit of reference documentation describing a single topic"[21].

By using this definition, we can avoid having to re-invent the wheel: clearly, the world does not need yet another DTD for man pages. Since XML is a subset of the *Standard Generalized Markup Language* (SGML), we do not need to change the projects objective; in fact, this will make it the more portable and useful!

"DocBook" itself is an SGML type definition used broadly in various Open Source: "The Linux Documentation Project" for example strongly recommends DocBook for writing a HOWTO. Various applications are able to read and write "DocBook" and conversion tools to other formats exist.

[2]You can download these and possible other older versions of the documentation and other relevant files from http://www.sourceforge.net/projects/mama/

Before we go on, we need to understand how “DocBook” in general and *RefEntry* in particular work, so it is recommended that the readers follow the link in the references to familiarize themselves with SGML and DocBook. A careful study of the examples given in [21] let us almost instantaneously create a template from which we will deduce precise examples. See Listing 1.

Listing 1: A SGML Template

```
<!DOCTYPE refentry PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<!--
  ‘‘MakeMan’’ SGML Template for a man page
  For more Information, please see http://mama.sourceforge.net

  $Author: Jan Schaumann <jschauma@netmeister.org> $
  $Id: man.template.sgml, v 0.2 2001/08/20 16:58:33 jschauma Exp $
-->

<refentry id="name">
  <refmeta>
    <refentrytitle>name.pl</refentrytitle>
    <manvolnum>section</manvolnum>
    <refmiscinfo class="date">Month Day, Year</refmiscinfo>
    <refmiscinfo class="source">Package</refmiscinfo>
    <refmiscinfo class="title">Title</refmiscinfo>
  </refmeta>

  <refnamediv>
    <refname>name</refname>
    <refpurpose>short description</refpurpose>
  </refnamediv>

  <refsynopsisdiv>
    <cmdsynopsis>
      <command>name</command>
        <arg choice="opt" rep="repeat">OPTION</arg>
        <arg choice="req">FILE</arg>
    </cmdsynopsis>
  </refsynopsisdiv>

  <refsect1>
    <title>DESCRIPTION</title>
    <para>
      Write a long description in here.  Do not explain the details of
      the options, which will be below, but the general usage.
    </para>
    <para>
      <command>name</command> is usually written in bold when mentioned,
      <emphasis>References</emphasis> in italics.
        </para>
  </refsect1>

  <refsect1>
```

```
    <title>USAGE</title>
    <para>
      Detailed usage.  If the program requires certain perparation
      before running, explain what needs to be done here.  Detailed
      explanation of unintuitive options may go here, as well.
    </para>
  </refsect1>

  <refsect1>
    <title>OPTIONS</title>
    <para>
      This program follows the usual GNU command line syntax,
            with long options starting with two dashes (‘--’). A
            summary of the options supported by <command>name</command>
            is included below.
    </para>
    <variablelist>
      <varlistentry>
        <term>-h, --help</term>
        <listitem>
          <para>Show summary of options and exit.</para>
        </listitem>
      </varlistentry>
      <varlistentry>
        <term>-v, --version</term>
        <listitem>
          <para>Show version information and exit.</para>
        </listitem>
      </varlistentry>
    </variablelist>
  </refsect1>

  <refsect1>
    <title>REQUIRES</title>
    <para>
      Prerequisites that need to be met for this program.
    </para>
  </refsect1>

  <refsect1>
    <title>VERSION</title>
    <para>
      Version Number
    </para>
  </refsect1>

  <refsect1>
    <title>BUGS</title>
    <para>
      Explain known bugs here.
    </para>
  </refsect1>

  <refsect1>
    <title>SEE ALSO</title>
    <para>
      <command>related(1)</command>
    </para>
    <para>
      <ulink url="http://somewhere.else">http://somewhere.else</ulink>
    </para>
  </refsect1>

  <refsect1>
    <title>AUTHOR</title>
    <para>
      <address>
        <firstname>First Name</firstname>
        <surname>Last Name</surname>
        <email>email@address.org</email>
      </address>
    </para>
  </refsect1>
</refentry>
```

3.1 Parsing SGML

Now that we have an SGML template, we can validate it. After all, the template won't do us any good if it is not valid SGML. *nsgmls*, part of the *SG*[24] package, is a command-line parser for SGML, which can be used to validate the example:

```
www:~/xml> nsgml -s man.template.sgml
```

If we do not get any error messages, we know that the file "man.template.sgml" contains valid SGML. For a detailed description of "nsgmls", please see *ns-gmls(1)*.

4 Coding Style

Before we start writing code, let us discuss a few issues that relate to the *Coding Style*. Since this is an Open Source project, not only can we *expect* this code to be read by others, we *want* it to be read by others. While in general, one should always write code as clean as humanly possible, we all know that people in general tend to take more precaution if they are aware that somebody else might see it. So, again, remember, your the code is *intended* for everybody to see and enhance.

To make it possible for others to understand the follow your code, a few common techniques should be obeyed by. The placement of braces, while often discussed and as lively a debate as the old "Vi-vs-Emacs" issue, is clearly nothing more than a matter of personal opinion. Therefor it does not make a difference if you write

```
if (x is true) {
  we do y
} else {
  we do z
}
```

or

```
if (x is true)
{
  we do y
}
else
{
  we do z
}
```

However, if you were to join an existing Open Source project, please adhere to the style used. Aside from that, there are some other issues that need to be paid attention to. An excellent summary of the most important issues are found in the Linux kernel documentation[25]. Please make sure to read and understand the document – while it is written with the Linux kernel code (C Code) in mind, it is applicable to other languages as well.

One thing that is not mentioned in this document but that is worth pointing out is the use of "$keyword: text $" strings for version control. There exists this nifty little tool *ident*, which can extract these from the input files, even if they are compiled binaries. While this originates from the *RCS* Revision Control System, it is very useful just by itself.

I have made a habit of including at least the following two strings in any file of a project:

```
$Author: Firstname Lastname <email@address> $
$Id: filename, v Version, YYYY/MM/DD HH:MM:SS who Exp $
```

By including these rather self-explanatory lines in each file of your code, it will later on be as easy as using `ident file` to determine the version of each file. While this may not necessarily strike you as the best thing since sliced bread for text-files (ie the source), it comes in handy when you want to check a binary. Refer to *ident(1)* for details.

5 Licensing

The license under which we wish to place the program is an important issue to ponder. While in general, I tend to favor the well-known GPL, in this case I decided to place the project under a BSD-license. The reasoning is very simple: the main purpose of the project is to promote the use and the creation of man pages for all applications/executables.

Listing 2: BSD Style License

```
Copyright (c) 2001, Jan Schaumann <jschauma@netmeister.org>
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:


* Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.
* The names of its contributors may not be used to endorse or promote
  products derived from this software without specific prior written
  permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

Should this project become a success and be widely used, it would be to the greater benefit of everybody if even developers of proprietary software could use the basis of this project to include and incorporate into their product.

Therefor, the disclaimer you see in Listing 2 will be in the “COPYING” file of the distribution as well as in each source-file.

6 “MakeMan” in Perl

6.1 Project Outline

Before we can start with anything else, we do need of course a project outline, an idea of how we want to progress. The above is not a bad start, but the project requires refinement. What exactly will “MakeMan” do and what should be the goal of the first release?

Obviously, the task at hand will involve a lot of text manipulation, so that it would seem reasonable to implement a first solution in *Perl*.

The first release shall be a program that takes an XML file as input and converts it into a valid man page, ie generate *roff* sources. As all good UNIX programs, the Perl implementation of “MakeMan” will operate upon *stdin* and *stdout* if the filenames are not specified through command-line options. Aside from some standard options such as *–help* and *–version*, this should suffice for a simple application such as this.

As mentioned above, SGML can be validated through the use of *nsgmls*, so we will make this tool a requirement for our program, as well as expect input from *stdin* to be the output of *nsgmls*. Should the user specify an input file, we can simply call *nsgmls*, so that these files may be `.sgml` files.

Let us now write down the specifications for the first release of the Perl implementation of “MakeMan”. While we’re at it, why not use the SGML template from above to:

1. show a practical example
2. show what the input and the output of “makeman.pl” should look like
3. define what “makeman.pl” actually does

Listing 3 shows the SGML file[3] describing the man page for “makeman.pl”, which ultimately we will include in the first release.

Listing 3: “MakeMan” man page in SGML

```
<!DOCTYPE refentry PUBLIC "-//OASIS//DTD DocBook V4.1//EN">

<!--
  ‘‘MakeMan’’ SGML Example Representation of a man page
  For more Information, please see http://mama.sourceforge.net

  $Author: Jan Schaumann <jschauma@netmeister.org> $
  $Id: makeman.pl.sgml, v 0.3 2001/08/27 13:52:57 jschauma Exp $
-->

<refentry id="makeman.pl">
  <refmeta>
    <refentrytitle>makeman.pl</refentrytitle>
    <manvolnum>1</manvolnum>
    <refmiscinfo class="date">August 27th, 2001</refmiscinfo>
    <refmiscinfo class="source">MakeMan</refmiscinfo>
    <refmiscinfo class="title">Writing Man Pages</refmiscinfo>
  </refmeta>

  <refnamediv>
    <refname>makeman.pl</refname>
```

[3]Note that you can download and browse these files from http://mama.sourceforge.net/.

```
  <refpurpose>parse an SGML man page into valid roff source</refpurpose>
</refnamediv>

<refsynopsisdiv>
  <cmdsynopsis>
    <command>makeman.pl</command>
      <arg choice="opt">-h</arg>
      <arg choice="opt">
        -i <arg choice="req">FILE</arg>
      </arg>
      <arg choice="opt">
        -o <arg choice="req">FILE</arg>
      </arg>
      <arg choice="opt">-v</arg>
  </cmdsynopsis>
</refsynopsisdiv>

<refsect1>
  <title>DESCRIPTION</title>
  <para>
    <command>makeman.pl</command> is part of
    <emphasis>MakeMan</emphasis>, a project to provide several
    frontends, GUI and non-GUI, to an SGML interface to write
    man pages.
      </para>
      <para>
          <command>makeman.pl</command>, written in Perl, parses an
    SGML input file and generates valid roff source that can be read
          by <emphasis>man</emphasis>.
  </para>
</refsect1>

<refsect1>
  <title>USAGE</title>
  <para>
    Per default, <command>makeman.pl</command> reads the output of
    <command>nsgmls</command> from <emphasis>stdin</emphasis> and
    write roff sources to <emphasis>stdout</emphasis>. Alternatively,
    the input- and output-files can be specified as command-line
    options, in which case the input file may be the SGML source.
  </para>
  <refsect2>
    <title>Supported Tags</title>
      <para>
        The following is a list of the tags supported by
        <command>makeman.pl</command>:
      </para>
      <variablelist>
        <varlistentry>
          <term><command>&lt;refentry&gt;</command></term>
          <listitem>
            <para>
              Encloses the entire document.
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;refmeta&gt;</command></term>
          <listitem>
            <para>
              Encloses the meta information. Together with
              <command>&lt;refentryterm&gt;</command>, <command>&lt;manvolnum&gt;</command> and
              <command>&lt;refmiscinfo&gt;</command> this section forms the
              Header of the man page.
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;refnamediv&gt;</command></term>
          <listitem>
            <para>
              Together with <command>&lt;refname&gt;</command> and
              <command>&lt;refpurpose&gt;</command> this forms the NAME section
              of the man page, which is used by
              <command>apropos(1)</command> and
              <command>whatis(1)</command>.
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;refsynopsisdiv&gt;</command></term>
          <listitem>
            <para>
              Encloses the SYNOPSIS section of the man page.
              In it, <command>&lt;cmdsynopsis&gt;</command> wraps the
              <command>&lt;command&gt;</command> and the optional <command>&lt;arg&gt;</command>
              tags.
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;refsect1&gt;</command></term>
          <listitem>
            <para>
              Encloses a main SECTION (".SH") in the man page.
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;refsect2&gt;</command></term>
          <listitem>
            <para>
              Encloses a SUBSECTION (".SS") in the man page.
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;title&gt;</command></term>
          <listitem>
            <para>
              Currently supported within the
              <command>&lt;refsect1&gt;</command> and  <command>&lt;refsect2&gt;</command>.
              Simiarly, <command>&lt;term&gt;</command> is used for lists
              (".TH").
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;para&gt;</command></term>
          <listitem>
            <para>
              Encloses a PARAGRAPH (".PP").
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;variablelist&gt;</command></term>
          <listitem>
            <para>
              Together with one or more <command>&lt;varlistentry&gt;</command>
              tags lists such as this can be created
              (".TP").
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;simplelist&gt;</command></term>
          <listitem>
            <para>
              A simple list.  Can have an attribute
              <emphasis>type</emphasis>:
              <simplelist type="vert">
                <member><emphasis>inline</emphasis> -
                comma separated</member>
                <member><emphasis>vert</emphasis> -- like
                this</member>
              </simplelist>
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;itemizedlist&gt;</command></term>
          <listitem>
            <para>
              An itemized list.  Can have an attribute
              <emphasis>spacing</emphasis> (if "compact" use
              as little whitespace as possible) and
              <emphasis>mark</emphasis>
              Within this list, the
              <command>&lt;listitem&gt;</command> tag can
              have an attribute
              <emphasis>override</emphasis>, which can have
              the same values as the
              <emphasis>mark</emphasis> attribute, causing
              the following items to be itemized by the
              according character.  Values for
              <emphasis>mark</emphasis> and
              <emphasis>override</emphasis> are:
              <simplelist type="vert">
                <member><emphasis>bullet</emphasis> - use
                \(bu to itemize</member>
                <member><emphasis>emdash</emphasis> - use
                \(em to itemize</member>
              </simplelist>
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;orderedlist&gt;</command></term>
          <listitem>
            <para>
              An ordered list.  Can have an attribute
              <emphasis>spacing</emphasis> (if "compact" use
              as little whitespace as possible),
              <emphasis>numeration</emphasis> and
              <emphasis>continuation</emphasis>.
              <emphasis>numeration</emphasis> can be one of
              "arabic" (default; enumerate  by incrementing
              arabic number), "loweralpha" (enumerate by
              incrementing the lowercase letters of the
              alphabet) or "upperalpha" (enumerate by
              incrementing the uppercase letters of the
              alphabet).
              <emphasis>continuation</emphasis> can be one
              of "restarts" (default; each ordered list has
              it’s own counting and starts from 0/a/A) or
              "continues" (each ordered list continues to
              increment the item numbers from where the last list
              stopped)
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;replaceable&gt;</command></term>
          <listitem>
            <para>
              Commonly used when listing the comand-line
              options, signifying that this parameter can be
              replaced.  It will be diplayed in italics
              ("\fI \fR").
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;filename&gt;</command></term>
          <listitem>
            <para>
              Encosing a fileame, which will be diplayed in italics
              ("\fI \fR").
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;optinal&gt;</command></term>
          <listitem>
            <para>
              Contents will be placed in square brackets
              ("[" and "]").
```

```
            </para>
          </listitem>
        </varlistentry>
        <varlistentry>
          <term><command>&lt;ulink&gt;</command></term>
          <listitem>
            <para>
              Similar to an HTML <command>&lt;a href&gt;</command> tag.
              Signifies that the contents are linked to a
              URL.  Will be displayed as "Contents (Link to
              \fIURL\fR)"
            </para>
          </listitem>
        </varlistentry>

      </variablelist>
    </refsect2>
  </refsect1>

  <refsect1>
    <title>OPTIONS</title>
    <para>
      A summary of the options supported by
      <command>makeman.pl</command> is included below.
    </para>
    <variablelist>
      <varlistentry>
        <term>-h</term>
        <listitem>
          <para>Show summary of options and exit.</para>
        </listitem>
      </varlistentry>
      <varlistentry>
        <term>-i <emphasis>FILE</emphasis></term>
        <listitem>
          <para>Read SGML from <emphasis>FILE</emphasis>.</para>
        </listitem>
      </varlistentry>
      <varlistentry>
        <term>-o <emphasis>FILE</emphasis></term>
        <listitem>
          <para>Write output to <emphasis>FILE</emphasis>.</para>
        </listitem>
      </varlistentry>
      <varlistentry>
        <term>-v</term>
        <listitem>
          <para>Show version information and exit.</para>
        </listitem>
      </varlistentry>
    </variablelist>
  </refsect1>

  <refsect1>
    <title>EXAMPLES</title>
    <variablelist>
      <varlistentry>
```

```
        <term>In a pipe:</term>
        <listitem>
          <para>
            <command>nsgmls</command>
            <emphasis>infile.sgml</emphasis> |
            <command>makeman.pl</command>
          </para>
        </listitem>
      </varlistentry>
      <varlistentry>
        <term>Alone:</term>
        <listitem>
          <para>
            <command>makeman.pl -i</command>
            <emphasis>infile.sgml</emphasis>
          </para>
        </listitem>
      </varlistentry>
    </variablelist>
  </refsect1>

  <refsect1>
    <title>REQUIRES</title>
    <para>
      Perl, SGMLSp, expat, nsgmls (part of SP)
    </para>
  </refsect1>

  <refsect1>
    <title>VERSION</title>
    <para>
      0.2
    </para>
  </refsect1>

  <refsect1>
    <title>BUGS</title>
    <para>
      Different indention levels within variable lists within
      subsections are not displayed correctly.
    </para>
    <para>
      Please report all bugs to the author.
    </para>
  </refsect1>

  <refsect1>
    <title>SEE ALSO</title>
    <para>
      <command>man(7)</command>
    </para>
    <para>
      <ulink url="http://mama.sourceforge.net">http://mama.sourceforge.net</ulink>
    </para>
  </refsect1>

  <refsect1>
```

```
    <title>AUTHOR</title>
    <para>
      Jan Schaumann <email>jschauma@netmeister.org</email>
    </para>
  </refsect1>
</refentry>
```

Let us now make sure that what we wrote down there is in fact a valid representation of a man page in SGML:

```
www:~/xml> nsgmls -s makeman.pl.sgml
```

Ok, everything looks peachy. Once we have run this input through our program "makeman.pl", we would like the output to look as in Listing 4.

Listing 4: "MakeMan" man page

```
.\"
.\" This page was created on 2001-08-28 17:27:01 by makeman.pl
.\" ‘‘makeman.pl’’ is part of the ‘‘MakeMan’’ project.
.\" For more information, please see http://mama.sourceforge.net
.\"
.TH makeman.pl 1 "August 27th, 2001" "MakeMan" "Writing Man Pages"

.SH NAME
makeman.pl \- parse an SGML man page into valid roff source

.SH SYNOPSIS
\fBmakeman.pl\fR
[ \fI\-h\fR ] [ \fI
\-i \fR\fIFILE\fR ] [ \fI
\-o \fR\fIFILE\fR ] [ \fI\-v\fR ]

.SH "DESCRIPTION"

.PP

\fBmakeman.pl\fR is part of
\fIMakeMan\fR, a project to provide several
frontends, GUI and non\-GUI, to an SGML interface to write
man pages.


.PP

\fBmakeman.pl\fR, written in Perl, parses an
SGML input file and generates valid roff source that can be read
by \fIman\fR.

.SH "USAGE"

.PP

Per default, \fBmakeman.pl\fR reads the output of
\fBnsgmls\fR from \fIstdin\fR and
```

```
write roff sources to \fIstdout\fR. Alternatively,
the input\- and output\-files can be specified as command\-line
options, in which case the input file may be the SGML source.

.SS "Supported Tags"

.PP

The following is a list of the tags supported by
\fBmakeman.pl\fR:

.\" Begin List
.TP
\fB<refentry>\fR

Encloses the entire document.

.TP
\fB<refmeta>\fR

Encloses the meta information. Together with
\fB<refentryterm>\fR, \fB<manvolnum>\fR and
\fB<refmiscinfo>\fR this section forms the
Header of the man page.

.TP
\fB<refnamediv>\fR

Together with \fB<refname>\fR and
\fB<refpurpose>\fR this forms the NAME section
of the man page, which is used by
\fBapropos(1)\fR and
\fBwhatis(1)\fR.

.TP
\fB<refsynopsisdiv>\fR

Encloses the SYNOPSIS section of the man page.
In it, \fB<cmdsynopsis>\fR wraps the
\fB<command>\fR and the optional \fB<arg>\fR
tags.

.TP
\fB<refsect1>\fR

Encloses a main SECTION (".SH") in the man page.

.TP
\fB<refsect2>\fR

Encloses a SUBSECTION (".SS") in the man page.

.TP
\fB<title>\fR

Currently supported within the
\fB<refsect1>\fR and  \fB<refsect2>\fR.
```

```
Simiarly, \fB<term>\fR is used for lists
(".TH").

.TP
\fB<para>\fR

Encloses a PARAGRAPH (".PP").

.TP
\fB<variablelist>\fR

Together with one or more \fB<varlistentry>\fR
tags lists such as this can be created
(".TP").

.TP
\fB<simplelist>\fR

A simple list.  Can have an attribute
\fItype\fR:

.br
\fIinline\fR \-
comma separated
.br
\fIvert\fR \-\- like
this

.TP
\fB<itemizedlist>\fR

An itemized list.  Can have an attribute
\fIspacing\fR (if "compact" use
as little whitespace as possible) and
\fImark\fR
Within this list, the
\fB<listitem>\fR tag can
have an attribute
\fIoverride\fR, which can have
the same values as the
\fImark\fR attribute, causing
the following items to be itemized by the
according character.  Values for
\fImark\fR and
\fIoverride\fR are:

.br
\fIbullet\fR \- use
\(bu to itemize
.br
\fIemdash\fR \- use
\(em to itemize

.TP
\fB<orderedlist>\fR

An ordered list.  Can have an attribute
```

```
\fIspacing\fR (if "compact" use
as little whitespace as possible),
\fInumeration\fR and
\fIcontinuation\fR.
\fInumeration\fR can be one of
"arabic" (default; enumerate  by incrementing
arabic number), "loweralpha" (enumerate by
incrementing the lowercase letters of the
alphabet) or "upperalpha" (enumerate by
incrementing the uppercase letters of the
alphabet).
\fIcontinuation\fR can be one
of "restarts" (default; each ordered list has
it’s own counting and starts from 0/a/A) or
"continues" (each ordered list continues to
increment the item numbers from where the last list
stopped)

.TP
\fB<replaceable>\fR

Commonly used when listing the comand\-line
options, signifying that this parameter can be
replaced.  It will be diplayed in italics
("\fI \fR").

.TP
\fB<filename>\fR

Encosing a fileame, which will be diplayed in italics
("\fI \fR").

.TP
\fB<optinal>\fR

Contents will be placed in square brackets
("[" and "]").

.TP
\fB<ulink>\fR

Similar to an HTML \fB<a href>\fR tag.
Signifies that the contents are linked to a
URL.  Will be displayed as "Contents (Link to
\fIURL\fR)"

.\" End List

.SH "OPTIONS"

.PP

A summary of the options supported by
\fBmakeman.pl\fR is included below.
.\" Begin List
.TP
\fB\-h\fR
```

```
Show summary of options and exit.
.TP
\fB\-i \fR\fIFILE\fR
Read SGML from \fIFILE\fR.
.TP
\fB\-o \fR\fIFILE\fR
Write output to \fIFILE\fR.
.TP
\fB\-v\fR
Show version information and exit.
.\" End List

.SH "EXAMPLES"
.\" Begin List
.TP
In a pipe:

\fBnsgmls\fR
\fIinfile.sgml\fR |
\fBmakeman.pl\fR

.TP
Alone:

\fBmakeman.pl \-i\fR
\fIinfile.sgml\fR

.\" End List

.SH "REQUIRES"

.PP

Perl, SGMLSp, expat, nsgmls (part of SP)

.SH "VERSION"

.PP

0.2

.SH "BUGS"

.PP

Different indention levels within variable lists within
subsections are not displayed correctly.

.PP

Please report all bugs to the author.

.SH "SEE ALSO"

.PP

\fBman(7)\fR
```

```
.PP
http://mama.sourceforge.net (Link to \fIhttp://mama.sourceforge.net\fR)
.SH "AUTHOR"
.PP
Jan Schaumann <jschauma@netmeister.org>
```

6.2 Coding Style Revisited: Perl Issues

When writing Perl, please make it habit to start each and every program with the following two lines:

```
/usr/bin/perl -w
use strict;
```

In addition, I would like to shamelessly copy, uhm, “cite” an excerpt from *http://www.perl.com/pub/a/2000/01/CodingStandards.html*:

1. The verbosity of all names should be proportional to the scope of their use
2. The plurality of a variable name should reflect the plurality of the data it contains. In Perl, $name is a single name, while @names is an array of names
3. In general, follow the language’s conventions in variable naming and other things. If the language uses variable_names_like_this, you should too. If it uses ThisKindOfName, follow that.
4. Failing that, use UPPER_CASE for globals, StudlyCaps for classes, and lower_case for most other things. Note the distinction between words by using either underscores or StudlyCaps.
5. Function or subroutine names should be verbs or verb clauses. It is unnecessary to start a function name with do_.
6. Filenames should contain underscores between words, except where they are executables in $PATH. Filenames should be all lower case, except for class files which maybe in StudlyCaps if the language’s common usage dictates it.

Furthermore, give `perldoc perlstyle` a careful read.

6.3 makeman.pl

Ok, now that we got all these formalities out of the way, let us write some code already! The first thing I usually start out with – well, the first thing *after* outlining the project and its specifications – is a skeleton that parses the command-line arguments and initializes a few global variables (if any). In this case, the simple function `init` as shown in Listing 5, will perform this task.

We know that we will deal with XML, so let's take a short trip over to CPAN[26] and investigate if some helpful modules might already be available. *XMLParser* sounds promising: "is an interface to James Clark's XML parser, *expat*". After installing *expat*[4], the usual routine installs the module:

```
tar zcvf XML-Parser-2.29.tar.gz
cd XML-Parser-2.29
perl Makefile.pl
make
make test
su -c "make install"
```

Further research via *http://www.google.com* reveals another interesting URL: *http://www.perlxml.com/faq/perl-xml-faq.html*, which suggests the *SGMLSpm* module, a "class library for parsing the output from James Clark's SGMLS and PSGMLS parsers." This is pretty much exactly what we need, so this gets installed right away as well.

After reading through `perldoc XML::Parser` and the documentation for *SGMLSpm* we realize that the latter will be fully sufficient for our project.

As mentioned above, if no `.sgml` file was specified on the command-line, we expect the input to be the output of *nsgmls*; otherwise, we call *nsgmls* ourselves on the given input file. The output of *nsgmls* is used to create a new object of type "SGMLS", which then can be analyzed by simply walking down the tree of elements. Depending on what kind of element we are dealing with, we call the appropriate subroutines. The function doing this work can be seen in Listing 6.

Whenever we encounter a new section, be it a main section ("Refsect1") or a subsection ("Refsect2", "Refsect3" etc.), we call the function "parseSection", which prints out the relevant information according to the tags encountered. This function needs to check a lot of conditions in a tedious way – if this reminds you of a compiler-class you took, don't be surprised. Excerpts of the function are shown in Listing 7.

[4] *apt-get install expat libexpat libexpat-dev* on a Debian machine does the trick, though installing from sources should not be a problem either

Listing 5: Parsing command-line options

```
sub init
{
  my %Options;
  my $ok = getopts('hi:o:v', \%Options);
  if (!$ok)
  {
    my $i;
    my @values = keys(%Options);
    foreach $i (@values)
    {
      if (!$Options{$i})
      {
        print "Option '$i' requires an argument.\n";
        exit(1);
      }
    }
  }

  if ($Options{'h'})
  {
    usage();
    exit 0;
  }
  if ($Options{'v'})
  {
    print "$NAME Version $VERSION\n";
    exit 0;
  }
  if ($Options{'i'})
  {
    $INSTREAM = $Options{'i'};
  }
  if ($Options{'o'})
  {
    $OUTSTREAM = $Options{'o'};
  }
}
```

Listing 6: Function “doParse”

```
.
.
.
    while ($event = $parse->next_event)
    {
        my $foo;
        if ($event->type eq 'start_element')
        {
            if ($event->data->name eq 'REFENTRY')
            {
                $start = 1;
                printHeaderComments();
            }

            if (!$start)
            {
                return 0;
            }

            if ($event->data->name eq 'REFMETA')
            {
                $event = parseMeta($parse, $event);
            }
            elsif ($event->data->name eq 'REFNAMEDIV')
            {
                $event = parseNameDiv($parse, $event);
            }
            elsif ($event->data->name eq 'REFSYNOPSISDIV')
            {
                $event = parseSynopsis($parse, $event);
            }
            elsif ($event->data->name eq 'REFSECT1')
            {
                print WRITE "\n.SH ";
                $event = parseSection($parse, $parse->next_event);
            }
        }

        if ($event->type eq 'conforming')
        {
            $valid = 1;
        }
    }

.
.
.
```

Listing 7: Function “parseSection”

```
.
.
.
        if ($type eq ’start_element’)
        {
            if ($data->name eq ’TITLE’)
            {
                printf WRITE "\"";
            }
            elsif ($data->name eq ’PARA’)
            {
                print WRITE "\n.PP";
            }
            elsif ($data->name eq ’COMMAND’)
            {
                printf WRITE "\\fB";
            }
.
.
.
            elsif ($data->name eq ’REFSECT2’)
            {
                print WRITE "\n.SS ";
                $event=parseSection($parse,$parse->next_event);
            }
        }
.
.
.
        elsif ($type eq ’end_element’)
        {
            if ( ($data->name eq ’REFSECT1’) ||
                    ($data->name eq ’REFSECT2’) ||
                    ($data->name eq ’REFSECT3’) )
            {
                return $event;
            }
            elsif ($data->name eq ’TITLE’)
            {
                print WRITE "\"\n";
            }
.
.
.
```

After testing our first implementation of “MakeMan” in Perl on a few SGML files, we convert the file `makeman.pl.sgml` (Listing 3) by issuing the following command:

```
./makeman.pl -i ../../xml/makeman.pl.sgml \
      -o ../doc/makeman.pl.1
```

Voilá – not so bad, I’d say. Eager to release this first version of “makeman.pl”, we can now start to create a package, write some accompanying documentation and then announce the software on the various websites so as to get some people to use it and find the bugs we overlooked. While some people believe that one should not release a program before version 1.0, I’m convinced it will help us keep our enthusiasm if we get feedback – no matter what kind – from other people. Finding and fixing bugs can only be done if the software is used, and very often it takes a fresh pair of eyes to realize the obvious that we might have overlooked. “Release early, release often!”

6.4 Packaging

In order to release the software, we will need to prepare a proper package; that is we need to write some documentation that will accompany the product, write installation instructions, copyright notices etc. We already have the man page available, so that we can now write the rest of the documentation.

I usually create a few HTML pages that cover the installation of the software and place that information into a plain ascii file as well, usually named “README” or ”INSTALL”. In order to get all these files installed properly, I usually provide a “Makefile” with a single target “install”. This is certainly not necessary for a small package such as this, but might prove convenient in the future.

With all this information, we can now create a directory structure as follows:

- makeman.pl-0.1
  - AUTHORS
  - CHANGES
  - COPYING
  - doc
    - html
      - index.html

            · index-1.html
            · index-2.html
            · index-3.html
            · index-4.html
            · index-5.html
        * makeman.pl.1
    - INSTALL
    - Makefile
    - README
    - src
        * makeman.pl

You will notice that we do provide all the files – AUTHORS, README etc – that one is used to see when downloading an Open Source package. Obeying this practice even for small packages makes it easier to maintain: for future releases, we will just need to modify the appropriate files.

Rolling a tarball is the first thing we do - after all, we want to distribute the package in it's most platform independent way. One `gzip`'ped and one `bzip2`'d tarball coming right up! Once we have created these, we can proceed to build Debian packages and RPM packages. While both these formats have their own quirks when building them there is excellent documentation available on the web. To build a `.deb` package, I usually follow the instructions given on [27]; for `.rpm`s, the "RPM-HOWTO" ([28]) comes in handy.

6.5 Releasing the package

Now that the packages have been built, we can announce the availability of our software on the web. First of all, we want to upload the files to our account at SourceForge. Just following the familiar procedure (uploading the files to upload.sourceforge.net and "Quick-releasing" from the Admin part of the website) we can release the packages easily.

Next we wish to announce the package on other free software sites such as Freshmeat (http://www.freshmeat.net). The procedure is pretty much the same for all the sites: one always has to specify the download location and a short descriptive blurb as well as a contact address.

Usually, I announce new software on the following sites:

- Freshmeat - http://www.freshmeat.net

- DaveCentral - http://linux.davecentral.com/
- IceWalkers - http://www.icewalkers.com/
- LinuxApps - http://www.linuxapps.com/
- Linuxberg - http://upload.tucows.com/linux.html

Depending on the site, it may take between a few hours and a few days for the packages to be announced.

6.6 Continued work on “makeman.pl”

After releasing the first package to the world, I have gotten some feedback from other developers; some feedback from other users and – fortunately – some bug reports. The continuing cycle of bug-fixes and releases of new versions has started. To document each and every one change of “makeman.pl” would be too tedious and certainly would be out of the scope of this document. However, as all changes are documented in the *CHANGES* file, the user will be able to easily follow the development of the software.

By the time of this writing, “makeman.pl” seems to work reasonably well. Quite a few man-pages (particularly for my other project “*The Missing Man Pages Project***??**) have been converted from SGML to roff-sources, and other people have shown some interest in the project.

My original goal of providing a better SGML-to-roff converter for man pages than docbook-to-man seems to have been reached, if only partially. Needless to say, development on this tool will continue as it is being used more and more.

New releases will be announced on websites supporting Free Software and on *http://mama.sourceforge.net*, of course. However, with the initial tool having reached a functional stage in the development cycle, we can now focus on providing more user-friendly front-ends that will eventually use the SGML format as their basis.

If you have any comments whatsoever related to “makeman.pl” (or the entire project, of course), please don’t hesitate to contact me at jschauma@netmeister.org. I will be more than happy to hear about bugs, so as to improve the software. Patches and other helpful suggestions will cause immense joy, as well.

References

[1] *XML*, http://www.w3.org/XML/

[2] *VI iMproved*, http://www.vim.org

[3] *Creating Automatic Configuration Scripts*, http://www.gnu.org/manual/autoconf/

[4] *A Program for Directing Recompilation*, http://www.gnu.org/manual/make-3.79.1/

[5] *Concurrent Versions System*, http://www.cvshome.org/

[6] *man*, http://www.gnu.org/directory/man.html

[7] *GNOME*, http://www.gnome.org

[8] *The Gimp Toolkit*, http://www.gtk.org

[9] *Qt*, http://www.trolltech.com

[10] *The K Desktop Environment*, http://www.kde.org

[11] *Java*, http://java.sun.com

[12] *Practical Extraction and Report Language*, http://www.perl.com

[13] *New curses*, http://www.gnu.org/software/ncurses/ncurses.html

[14] *Python*, http://www.pythonlabs.com/

[15] *Fast Lexical Analyzer Generator*, http://www.gnu.org/software/flex/flex.html

[16] *Bison*, http://www.gnu.org/software/bison/bison.html

[17] *PHP*, http://www.php.net/

[18] *Simple Object Access Protocol (SOAP)*, http://www.w3.org/TR/SOAP/

[19] *SourceForge.net* http://www.sourceforge.net

[20] *DocBook: The Definitive Guide* http://www.docbook.org/tdg/en/html/docbook.html

[21] *DocBook Element Reference: refentry* http://www.docbook.org/tdg/en/html/refentry.html

[22] *Standard Generalized Markup Language* http://www.w3.org/MarkUp/SGML/

[23] *The Linux Documentation Project* http://www.linuxdoc.org

[24] *SP* http://www.jclark.com/

[25] *Linux kernel coding style* /usr/src/linux/Documentation/CodingStyle, or http://puffin.external.hp.com/lxr/source/linux-2.3/Documentation/CodingStyle

[26] *Comprehensive Perl Archive Network* http://www.cpan.org

[27] *Making A Debian Package* http://people.debian.org/ jald-har/make_package2.html

[28] *RPM-HOWTO* http://www.linuxdoc.org/HOWTO/RPM-HOWTO/index.html

[29] *The Missing Man Pages Project* http://www.netmeister.org/misc/m2p2/index.html