`sdc` – Sgml Document Compiler

Jörg Wittenberger

TUD IBDR

Preface

This document is ment to hold the user documentation of `sdc` in an extensive way.

It is assembled from some subdocuments, which already share some text parts. If you find something repeated – you know why now.

Chapter 1 is simply the manual page of `sdc`.

For an short overview see the intro, which is Chapter 2 here.

Contents

Chapter 1. sdc – an extensible SGML formatter

1.1. SYNOPSIS

`sdc` `-o` *outputfile* [`-O` *format*] [`-L` *directory*] [`-i` *directory*] [`-m` *file*] [`-R` *startup file*] [`-V` *level*] *files ...*

1.2. DESCRIPTION

Typeset is an extensible formatter for documents. It transforms documents using SGML markup into various target formats.

Typeset comes with a couple of document type definitions (DTD's).

The DTD's feature the reuse of text, minimization of markup and readability of the SGML source. They share their elements as much as possible.

The formatting differs due to the features possible in the target format and to the rules common for the type of the document. This includes the automated rearrangment of text and insertion of standard parts like contents sections, sorted index and bibliography. The latter for instance is composed from the items of a database which are referenced in the document. For some formats the output may be spread over a couple of files. See the target type documentation for details.

According to the goal of text reuse and the aim to support many target formats, these DTD's don't attempt to cover each and every case possible. Instead, they try to provide all elements nessesary for daily use and leave the implementation of special features to extensions.

It is also possible to have parts of the documents using other notations. E.g., pictures drawn with tgif, xfig, the @Fig package of Lout or encapsulated postscript.

It is fairly easy to coerce sdc to parse documents with other DTD's. But this implies to write rules for formatting in the desired target format(s), or fit in another parsing stage which changes it into a form as if it was marked acording to a supported DTD.

The transformation (formating) is described by files of scheme code related to both, the document type and the target format. Only combinations of common value are supported by default. (For instance for letters only PostSript output is defined.)

1.3. OPTIONS

`-o` *filename* Set the name for the output file. If omitted or set to − the output goes to the standard output. (This can cause problems with some target formats if they split the document.)

Some target formats (e. g., HTML) hard wire the given name of the output file into the file(s) created (for cross referencing). Though it's wise not to give a path including a directory but only a file name for the output file.

`-O` *type* Set the target format. *type* can be:

- `ps` create a PostScript document.
- `latex` create a LaTeX file.
- `html` create a HTML page.
- `info` create an Info file.
- `man` create a man page.

`literate` create source files (literate programing)

`rtf` (only partially supported) create a RTF file.

`slide` create a PostScript file holding the slides from the document.

If the `-O` switch is omitted a guess is made from the extension of the output file name. If neither gives a target type this is an error.

`-D` *directory* Add *directory* in front of the path searched for entities (files) of the docuemnts. Each option can add only one directory. Multiple options are processed left to right, i. e., the last directory at the command line is searched first.

`-i` *entityname* Ensure, that a definition like

```
<!ENTITY % entityname "INCLUDE" >
```

precedes the processing of the documents. This is useful for optional including of marked sections. Refer to the manual sgmls(1) for a detailed description. This option is passed to sgmls.

`-m` *file* Extend the list of *catalog files* to search for some SGML entities. Refer to the manual sgmls(1) for a detailed description. This option is passed to sgmls.

`-L` *dirname* Set the name of the directory to use as library of files to search for target format descriptions.

`-R`*file* Set a startup file to load after the default `/.typesetrc`. Multiple `-R` options are allowed and processed in the given order. The files argument is treated to be either a path name to the file or one relative to the `rc` directory of a directory in the library (see `-L`).

Startup files can have their own arguments. If the argument given with a -R option contains a colon, only the half up to that colon gives the file name to be loaded. The rest of the argument (without the colon) is assigned to the variable `*-R-option-argument*` while the specified file is loaded. If there was no colon in the argument, `#f` is assigned.

`-V` *level* Be verbose and don't delete temporary files (for debuging). *Level* must be a number. The default for *level* is 1. This will give only warnings and (for a historical reason) a message upon success. Higher values give more messages.

With the `-R` option there are additional (long) options available to change the over all behavior. These are used by supplying one or more of the following file names to the `-R` option.

nidx Pretend having the `NIDX` token in the face attribute.

1c, 2c, 1s, 2s Simmilar to `nidx` modify the value in effect of the face attribute in the top level document.

no-margin No page margin in the (ascii) output. Only implemented for lout processing at the moment.

HTML2 Don't use HTML-3 features in formatting.

linuxdoc Work around some hard wired assumtions of the linuxdoc DTD when processing.

Attention! Be careful to supply the *exact* name to the `-R` option. The same policy as for dotfiles applies to those files: if they don't exist they are silently not loaded! THere is no warning message.

1.4. RETURN CODE

sdc returns an exist value of 0 upon successful SGML parsing and 1 if the input does not conform to the DTD. Semantic errors like unresolved references are currently not covered.

1.5. EXAMPLES

Set the environment variable `DOCPATH` to a usefull value like:

```
setenv DOCPATH $HOME/text/
```

Call sdc to generate a html page from text.sgml:

```
sdc -o text.html text.sgml
```

1.6. ENVIRONMENT

Typeset recognizes the following environment variables:

`DOCPATH` This path is used to find the entities of the document. It gets extended (at the end) by sdc to include the files of it's own. Also directories give by a `-D` option are prepended.

Usually a good value for `DOCPATH` is something like `$HOME` or `$HOME/text:$HOME/doc`.

`SGML_CATALOG_FILES` The files mentioned by this variable are consulted by the underlying parser to find some SGML entities. For a detailed description refer to the the manual sgmls(1) . This variable gets extended by sdc to include one file of its own, the first file named `CATALOG` found in the library. As for sgmls the value can be extended by the `-m` option, which is simply passed to sgmls.

Usually it's good to leave this variable alone.

`TYPESETLIB` This varaible is used by sdc to find the directories to search for formatting translation files and the DTD's and CATALOG files for the underlying SGML parser. It may point to one directory or a list of directories seperated by colons. This value can be overwritten by the `-L` option.

Usually it's good to leave this variable alone, except if you want to overwrite some but not all files of the library.

1.7. FILES

`personal.data` is used by the DTD's which come with sdc to find definitions for the SGML entities related to the author. These two are myself and my-Inst. It may define some more. But these are used to insert default values for the name and the institution of the author. Therefore it's a good idea to set the environment variable `DOCPATH` so sdc will find this file. An example how to set up the content of this file comes with sdc.

`/.typesetrc` if any, is loaded after startup and comand line evaluation. It might contain any scheme code.

sdc uses the files and directory structure in its library to parse the document and determine the formatting. For a descrition of this refer to the developers documentation.

Furthermore sdc uses the *current working directory* when writing temporary files. (See the developers documentation for reasoning.) Make sure to change into a writable directory before invoking sdc.

1.8. CONFORMING TO

The underlying sgmls conforms to ISO 8879. Therefore sdc too.

1.9. NOTES

At the moment there is no complete developers documentation available. Support for RTF output is still far from complete and uses the old mechanism.

1.10. DIAGNOSTICS

An error

`SGML error at`*file*`, line`*number*

comes from the underlying SGML parser and indicates text not conforming to the document type definition. Usually this will imply a lot of other errors.

```
*** ERROR:bigloo:eval:
```

Indicates a bug in the format translation files.

1.11. AUTHOR

This was written by Jörg Wittenberger.

1.12. SEE ALSO

sgmls(1) , lout(1) , info(5) , latex(1) , xfig(1) .

Chapter 2. Typeset, a short introduction

This is a short introduction into `sdc`.

`sdc` is a formater for SGML documents.

This document describes the use of the document types which are currently handled from `sdc`. It does *not* describe the adaption of sdc to other document types or target formats.

2.1. Overview

Typeset is an extensible formatter for documents. It transforms documents using SGML markup into various target formats.

Typeset comes with a couple of document type definitions (DTD's).

The DTD's feature the reuse of text, minimization of markup and readability of the SGML source. They share their elements as much as possible.

The formatting differs due to the features possible in the target format and to the rules common for the type of the document. This includes the automated rearrangment of text and insertion of standard parts like contents sections, sorted index and bibliography. The latter for instance is composed from the items of a database which are referenced in the document. For some formats the output may be spread over a couple of files. See the target type documentation for details.

According to the goal of text reuse and the aim to support many target formats, these DTD's don't attempt to cover each and every case possible. Instead, they try to provide all elements nessesary for daily use and leave the implementation of special features to extensions.

It is also possible to have parts of the documents using other notations. E.g., pictures drawn with tgif, xfig, the @Fig package of Lout or encapsulated postscript.

It is fairly easy to coerce sdc to parse documents with other DTD's. But this implies to write rules for formatting in the desired target format(s), or fit in another parsing stage which changes it into a form as if it was marked acording to a supported DTD.

The transformation (formating) is described by files of scheme code related to both, the document type and the target format. Only combinations of common value are supported by default. (For instance for letters only PostSript output is defined.)

Currently there are these DTD's

document Simple "plain" documents.

report Technical reports, documentation etc.

book Books (longer documentations).

bibdata Bibliography database.

manpage Pages for the Unix(TM) man command.

brief A letter according to DIN.

Currently the following target formats are supported:

- PostScript (for english and german text)

- LaTeX
- HTML (Hyper Text Markup Language)
- Info (to be used into the on-line help of `emacs`)
- man suitable for `roff -man`
- ASCII
- source code (literate programing)
- slide to extract slides from a document
- limitted support for RTF

Future output formats will include: roff -ms (or -mm), RTF.

2.2. What is SGML

This section may be translated some time. It's present in the german version and it's intended to explain the advantages of text processing over word processing and the advantages of a generalized markup over target dependand one.

If you retrieved the package you are probably convinced anyway. So it's obsolete.

You may refer for a description of SGML to [1].

2.3. Invocation

First of all set your environment variable `DOCPATH`. It should point to the top directory of the tree, where all your files containing SGML data are stored.

Setting this variable could be done like this (for (t)csh) and compatible shells.

```
setenv DOCPATH $HOME
```

Invocation systax for sdc:

`sdc` [*options*|*filename.sgml*]*

Options:

`-o` *filename* Set the name for the output file. If omitted or set to − the output goes to the standard output. (This can cause problems with some target formats if they split the document.)

Some target formats (e. g., HTML) hard wire the given name of the output file into the file(s) created (for cross referencing). Though it's wise not to give a path including a directory but only a file name for the output file.

`-O` *type* Set the target format. *type* can be:

`ps` create a PostScript document.

`latex` create a LaTeX file.

`html` create a HTML page.

`info` create an Info file.

`man` create a man page.

`literate` create source files (literate programing)

`rtf` (only partially supported) create a RTF file.

`slide` create a PostScript file holding the slides from the document.

If the `-O` switch is omitted a guess is made from the extension of the output file name. If neither gives a target type this is an error.

`-D` *directory* Add *directory* in front of the path searched for entities (files) of the docuemnts. Each option can add only one directory. Multiple options are processed left to right, i. e., the last directory at the command line is searched first.

`-i` *entityname* Ensure, that a definition like

```
<!ENTITY % entityname "INCLUDE" >
```

precedes the processing of the documents. This is useful for optional including of marked sections. Refer to the manual sgmls(1) for a detailed description. This option is passed to sgmls.

`-m` *file* Extend the list of *catalog files* to search for some SGML entities. Refer to the manual sgmls(1) for a detailed description. This option is passed to sgmls.

`-L` *dirname* Set the name of the directory to use as library of files to search for target format descriptions.

`-R` *file* Set a startup file to load after the default `/.typesetrc`. Multiple `-R` options are allowed and processed in the given order. The files argument is treated to be either a path name to the file or one relative to the `rc` directory of a directory in the library (see `-L`).

Startup files can have their own arguments. If the argument given with a -R option contains a colon, only the half up to that colon gives the file name to be loaded. The rest of the argument (without the colon) is assigned to the variable `*-R-option-argument*` while the specified file is loaded. If there was no colon in the argument, `#f` is assigned.

`-V` *level* Be verbose and don't delete temporary files (for debuging). *Level* must be a number. The default for *level* is 1. This will give only warnings and (for a historical reason) a message upon success. Higher values give more messages.

With the `-R` option there are additional (long) options available to change the over all behavior. These are used by supplying one or more of the following file names to the `-R` option.

nidx Pretend having the `NIDX` token in the face attribute.

1c, 2c, 1s, 2s Simmilar to `nidx` modify the value in effect of the face attribute in the top level document.

no-margin No page margin in the (ascii) output. Only implemented for lout processing at the moment.

HTML2 Don't use HTML-3 features in formatting.

linuxdoc Work around some hard wired assumtions of the linuxdoc DTD when processing.

Attention! Be careful to supply the *exact* name to the `-R` option. The same policy as for dotfiles applies to those files: if they don't exist they are silently not loaded! THere is no warning message.

A typical call would be:

```
sdc -o text.ps source.sgml
```

2.3.1. Environment

Typeset recognizes the following environment variables:

`DOCPATH` This path is used to find the entities of the document. It gets extended (at the end) by sdc to include the files of it's own. Also directories give by a `-D` option are prepended.

Usually a good value for `DOCPATH` is something like `$HOME` or `$HOME/text:$HOME/doc`.

`SGML_CATALOG_FILES` The files mentioned by this variable are consulted by the underlying parser to find some SGML entities. For a detailed description refer to the the manual sgmls(1) . This variable gets extended by sdc to include one file of its own, the first file named `CATALOG` found in the library. As for sgmls the value can be extended by the `-m` option, which is simply passed to sgmls.

Usually it's good to leave this variable alone.

`TYPESETLIB` This varaible is used by sdc to find the directories to search for formatting translation files and the

DTD's and CATALOG files for the underlying SGML parser. It may point to one directory or a list of directories seperated by colons. This value can be overwritten by the `-L` option.

Usually it's good to leave this variable alone, except if you want to overwrite some but not all files of the library.

2.3.2. Files

`personal.data` is used by the DTD's which come with sdc to find definitions for the SGML entities related to the author. These two are myself and my-Inst. It may define some more. But these are used to insert default values for the name and the institution of the author. Therefore it's a good idea to set the environment variable `DOCPATH` so sdc will find this file. An example how to set up the content of this file comes with sdc.

`/.typesetrc` if any, is loaded after startup and comand line evaluation. It might contain any scheme code.

sdc uses the files and directory structure in its library to parse the document and determine the formatting. For a descrition of this refer to the developers documentation.

Furthermore sdc uses the *current working directory* when writing temporary files. (See the developers documentation for reasoning.) Make sure to change into a writable directory before invoking sdc.

2.4. Document types

2.4.1. Document type `document`

We start with an example:

```
<!doctype document public "-//JFW//DTD Document//EN" >
<document>The Title
<sect>Intro

Here goes the introductory text.
<sect>We continue

This is the text body of the first section. It's going to be a little
bit longer to show, that the formating of the source file really
doesn't matter for the output.

We start a new paragraph simply by inserting a newline.
```

A document starts as every SGML document with the document type declaration. It's opened with the document tag.

There are the following attributes available to a document:

date The date of the document.

author The author.

inst The institution.

lang The lang attribute for the document is obsolete. It has the same effect as changing the public document language in the document declaration.

face This attribut is intended to affect the representation of the output. E. g., a value of `1c` should cause printing in one column and a value of `2c` should result in two column. A token `nidx` will supress the generation of an index even if `index` tags are used. For `face` multiple values can be assigned (in quotes). See documentation for the target formats for the treatment of this argument.

Then some paragraphs might follow. Next after these paragraphs either none or more than one sections can follow. Eventually and appendix can finish the document.

2.4.2. Dokument type `report`

A technical report consists of an abstract followed by sections and possible appendixes. Appendices themself are divided into sections.

Again an example:

```
<!doctype report public "-//JFW//DTD Report//EN">

<report
date="Today"
>Reporttitel

<abstract>

This is the abstract.
</abstract>

Here comes some introductory text. For some targets (e.g., PostScript
because Lout doen't allow text at this point) this text is taken to
be and section named "Introduction" (in the document language).

<sect>first section

The text.
<sect>second section

More text.
<appendix>
<sect>Appendix

Text of the appendix.
```

The part from and including the `appendix`-tag is optional.

The sections may be divided by `sect1`-Tags. These by `sect2`. There is no division of the `sect2`.

2.4.3. Dokument type `book`

Books are written using a document type declaration like:

```
<!doctype book public "-//JFW//DTD Book//EN">

<book
date="Today"
>Title of the Book
```

The `<book>` tag has the same attributes as the document or report tag. This is for consitency but questionable. The `inst` attribute would better be called publisher. Future versions will eventually rename this attribute.

In principle a book is divided into chapters (at least two) with the `<chapt>` tag as a simple document is divided into sections. Chapters themself consist of sections. Prior to the first chapter there may be two special sections named `<preface>` and `<intro>`.

Future versions will support grouping of chapters into `<part>`'s.

2.4.4. Document type `manpage`

The document type `manpage` is intended to produce pages for the Unix man command. It restricts the set of available elements to what `roff -man` can handle. Especially figures are not valid elements. Furthermore this document type enforces a “good style” for the man page. The possible sections are predefined. (Well there is a backdoor to use self defined sections, see `doc/badman.sgml`.)

A manpage begins like this:

```
<!doctype manpage public "-//JFW//DTD Manpage//EN">

<manpage
date="today"
>
TYPESET
<short>
an extensible SGML formatter
<synopsis> <code/typeset/ <var/options/ <var/files/ ...
<descript>
```

Up to this point the elements are enforced. That is, you can’t write a man page without a title (between the `<manpage>` and the `<short>` tag, a short description and a synopsis. Up to the synopsis these elements can’t even spawn multible lines, which couldn’t be formatted to be a man page.

The description section can be preceded by a config section. But it while config can be omitted `<descript>` can not.

The description section plays a special role in another way: it’s the only one which may be devided into subsections using `<sect1>`.

The following sections are also valid for a man page. They must appear in this order:

`options, return, errors, example, env, files,conform, notes, diag, restrict, history, see.`

Here what goes into the sections:

2.4.4.1 Synopsis

Tag: `<synopsis>`.

The command or what to write to call the funtion. For example:

```
<code/typeset/ <var/options/ <var/files/ ...
```

or for C funtions and system calls:

```
#include <something.h>
int foo(int bar);

int foo2(int bar);
```

2.4.4.2 Config

Tag: `<config>`.

This section explains how a device is configured: major/minor numbers, their meanings and the meaning of the device name.

2.4.4.3 Description

Tag: `<descript>`.

long drawn out discussion of the program. It’s a good idea to break this up into subsections. Using `<sect1`.

There is no BUGS section, instead discuss them here.

2.4.4.4 Options

Tag: <options>.

Some people make this separate from the description.

It's intended to hold one <desc> element 2.5.3. Don't forget to use the elements <code> and <var> here. Example:

```
<desc>
<dt><code/-option/ <var/file/

    Text describing the option.
...
```

2.4.4.5 Return Value

Tag: <return>.

What the program or function returns if successful.

2.4.4.6 Errors

Tag: <errors>.

Return codes, either exit status or errno settings.

2.4.4.7 Examples

give some example uses of the program

2.4.4.8 Environment

Tag: <env>.

Environment Variables this program might take care about

2.4.4.9 Files

Tag: <files>.

All files used by the program. Typical usage is a <desc again. 2.5.3

2.4.4.10 Conforming To

Tag: <conform>.

SVID [EXT], AT&T, POSIX, X/OPEN, BSD 4.3

2.4.4.11 Notes

Tag: <notes>.

Miscellaneous commentary

2.4.4.12 Diagnostics

Tag: <diag>.

all the possible error messages the program can print out, and what they mean.

2.4.4.13 Restrictions

Tag: <restrict>.

bugs you don't plan to fix :-)

2.4.4.14 History

Tag: `<history>`.

Programs derived from other sources sometimes have this.

2.4.4.15 See Also

Tag: `<see>`.

Other man pages to check out, like:

```
<ref t=m id="man(1)"//, <ref t=m id="man(7)"//, <ref t=m
id="makewhatis(8)"//.
```

Refer to 2.5.5 for a detailed description of `<ref>`.

2.4.5. Document type `brief`

The Brief is a DTD according to the German DIN standard. Example:

```
<!doctype brief public "-//JFW//DTD Brief//DE">
<brief fenster=ja>
<von>&my-adr;
<an>
<adr
NAME="Mustermann"
VORNAME="Erwin"
ORT=Musterhausen
STRASSE="Musterstra&ss;e 7m"
PLZ="01000">
<datum>1. Januar 1995
<betr>Musterschreiben
<anrede>Lieber Erwin,
<text>
Heute reden wir &ue;ber Musterbriefe.

Wie gef&ae;llt Dir das?
<gruss>Ciao
<anlage><pkt>1 Musterschreiben
```

This gives a letter according to the DIN. The sender address appears a second time in the window of the envelope. Also fold marks are printed `fenster=nein` will suppress this.

The tags `anrede, gruss` and `anlage` may be omitted. In this case standard text is inserted for the opening and closing. Enclosed is omitted.

The form of the address looks a little complicated. This is because it’s intended to come from a database. If you set up on, its use will look like the from address (`<von>...`).

The entity `my-adr` is defined in the file holding the personal date described in section 2.10.

2.5. Elements of the document

The different document types share the same elements (Except for “brief”).

2.5.1. Paragraphs

Paragraphs are seperated with a <p>-Tag.

To reduce markup, an empty line counts as an <p>-tag. Sequences of those tags are reduced to just one.

2.5.2. Enumerations

There are two kinds of lists, ordered and unordered.

<enum> opens an enumerated list.

<list> opens a not numbered list.

<item> starts a new item.

<o> same as <item> (short form).

Both `<list>` and `<enum>` have to be closed (by `</list>` or `</enum>`).

Example:

```
<list>
<item> Language to describe the logical structure of text.
<o> a tool and library to format SGML text into
<enum>
<o>PostScript
<item>HTML
</enum>
<item>One more point
</list>
```

formats to:

- Language to describe the logical structure of text.
- a tool and library to format SGML text into
    1 PostScript
    2 HTML
- One more point

2.5.3. Glossare

Glossares are declared by the `<desc>` Tag.

Again an example:

```
<desc

<dt/<desc>/ opens a description

<dt/<dt>/ encloses the described topic

</desc>
```

And the corresponding output.

<desc> opens a description

<dt> encloses the described topic

You don't want to put newlines (starting paragraphs) between the `<desc>` and the `<dt>`. If you want (as me) omit the closing > from the `<desc>` (say `<desc`).

2.5.4. Pictures

We don't have a chance to describe pictures in terms of SGML, but we can tell where the formating application should put them.

To include pictures one can use:

- Entity definitions and references. The reference is either a less controlled entity reference like `&`*name*`;` or it's done by using the `foreign`-tag. The latter should be prefered, because it could serve more control, but in fact there not much of a difference.

  This is the prefered method.
- Inlined code, see 2.5.13.
- For some cases, like including GIF-pictures into HTML documents, it's nessesarry to compromise the portability and target independance of the document.

  Those cases need processing instructions. For the mentioned case one needs the following:

  1 A `SDATA` entity to get a literal > into the final output. This could be achieved by having this definition in the "header":

  ```
  <!ENTITY lit-gt SDATA ">" >
  ```

  2 A Processing instruction at the place where the picture (here *picture.gif*) goes:

  ```
          <? <IMG SRC=picture.gif> &lit-gt;
  ```

  (We need the `lit-gt` because SGML has no way to have (escape) a >-character within a processing instruction)

In all the cases there are two ways how to include the picture, either directly between the lines of the text or as a floating part of the document. The latter form will allow you to put a caption line on it and an identifier to refer to from any part of the document.

Having pictures between the lines usually doesn't look very professional. For the effort word processors require to handle floats, it became common to do so. You should think twice whether this form is appropriate in your case.

If you use entities and notations for your pictures, you need to declare both in the header of your document. See 2.17 for a more detailed description and refer to [1] for full details.

The local installation2.18 may provide some predefined notations. A full installation supports at least eps, fig, lfig, roff, latex, tgif.

To include a picture entity (with entity name "name") defined using notations between the lines of text just write the entity reference like this: `&name;`.

To include the same picture as a floating object use the `<figure>` tag:

```
<figure id=refname>
<foreign file=name>
<caption/The caption line/
```

The `id` attribute tells the name to be used by cross references2.5.5 for this figure.

2.5.5. Cross References

Cross references are introduced by the `ref`-tag. It has two attributes:

id The ID it refers to. The interpretation of the id depends on the value of the **t** attribute.

t The type of reference. There are 3 possible values:

X (the default) The ID refers to some id in the same document.

Note that this mean a document as SGML understands it. If you are inside something which is a full document by itself, but is included as a `SUBDOC` Entity into some other document, you can refer to any id within this document, but *not* any in the outer document!

B Bibliography reference. The Id notes the `Tag` Attribute value in the database 2.7.

M The ID refers to a manpage.

U A URL. The ID hold the complete URL.

2.5.6. Emphasize

To emphasize long parts of text there is the `<quote>`-tag.

Example:

```
<quote/Important result: long citations, definitions and other
material which should be emphasized is enclosed by quote-tags./
```

And the result:

> Important result: long citations, definitions and other material which should be emphasized is enclosed by quote-tags.

The `<quote>`-tag has a `style` attribute accepting the values `default` (wich is the same as if nothing is given) and `center`. These give the recomented style. Default is to narrow the text a little, while center narrows and centers the quoted material.

2.5.7. Footnotes

Footnotes are enclosed by `<footnote>`-tags.

```
<footnote/You can't have figures in footnotes./
```

Will format as[1].

2.5.8. Notes

Sometimes you may like to have longer side notes. These are enclosed by `<note>`. This looks like this:

This text has been enclosed by `<note>` and `</note>`.

2.5.9. Verbatim copied text

For excerpts of source code there is the `<verb>`-tag. The examples in this text are made mostly by this. Code inside of the verb marked region is not at all processed for SGML references. Therefore no references to any entity are possible inside (like `<`).

There is also a variant of `<verb>` called `<rverb>`. This means "replacable" verbatim. The contents of `<rverb` region is parsed for entity references. Thus allowing references to external entities or SGML-end tags inside. The examples which have end tags inside are written using this element.

2.5.10. Emphasizing words

To emphasize short phrases or words you can choose from the prefered tag `<em>` and these:

<em> This will produce a different kind of emphasizing due to the level of use. (slanted, bold..)

<strong> **Heavy** emphasize.

<bf> **Bold face**

<it> *italic*

<tt> `tele type` kind

2.5.11. Linguistic Markup

For documentation purpose it is common to distinguish between literals (code), variables and meta characters. Therefore markup exists. These tag don't nest, that means each of them ends each other of this group.

---

[1]You can't have figures in footnotes.

For literals use `<code>`, variables `<var>` and for meta characters `<meta>`.

Please don't look for use of them in this manual, they are late introduced.

2.5.12. explicit line and page breaks

In rare cases you might need to insert unconditional newlines. There is the general entity `nl` for.

To enforce a page break at a certain position use the `<newpage>` element.

Don't use it too much. It might look strange in some output formats. You'll need it most with the `slide` target, so best enclose them all the time with a marked section like:

```
<[ %Slide [ <newpage> ]]>
```

2.5.13. Inline Code

Using the `<inline>` element you can include code using other notations (see 2.17) as available local at your site (see 2.18).

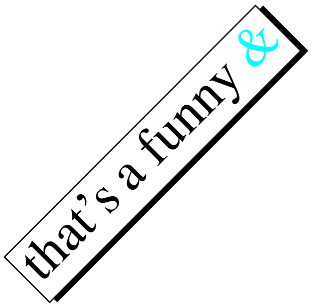
The `<inline>` element takes one argument `n` which must be assigned to a valid notation. This way you can achieve special effects or write tables and equations.

Inside the `<inline>` element you can't write other SGML markup, but you can refer to predefined entities (using the `&name;` notation). This raises the question how to include a & followed by a letter. For this purpose you need to write a decimal character reference i.e., `&`.

Example:

```
<inline n=lout>
|1rt 45d @Rotate @ShadowBox 2f @Font {that's a funny cyan @Color "&" }
</inline>
```

will give you



If you use the `<inline>` feature, be careful about the filenames you use: sdc will eventually (due to the need of the target format) create files matching the pattern:

basename-of-output−number`.`*extension*.

For admins: The example above might not work "out of the box" at your site since it uses a notation which needs to be set up. See 2.18and the documentation about target formats for things nessesary.

If you're going to install sdc. See the file `doc/notations.sgml` of the installation and adapt the it to your needs.

2.5.14. Tables

In response to various requests for tables one or more the following "syntax", or to put it better the following ideas will be implemented. I strongly encourage everybody to mail me comments about this.

Currently there is one SGML construct to describe tables within sdc. If someone comes up with a better solution for (it shouldn't look too strange in the source – sdc is supposed to remain a "don't worry" application) it will be incorporated.

A table is enclosed with the <table> tag. It can contain one of the implemented kinds of table (currently only <tbl>).

2.5.14.1 tbl

A tbl-table consists of patterns and rows. Each row must name the pattern it is formed after or otherwise it uses the most recent used or defined pattern. A row itself is a sequence of cells. Cells begin either with a <c>-tag or simply a |.

A pattern consists of tags describing the alignment. There are left, right, center, decimal and block. Between these <sep> are allowed to request vertical lines in the table.

The formatting of the cells within a row is described by the associated tag in the used pattern.

Probably an example is the better way to explain.

The implementation of tables is not finished.

The here defined syntax has a) not all the features of all the backends, so you have to drop to plain backend (e.a., inline) code ifyou really need those features b) it has more features than all backends have in common, therefore depending on the backend some features are silently dropped off.

For HTML HTML-3 Tables are implemented at the moment. As there is virtually no client to display these, one can change to use fixed font preformatted tables instead. (If you have a client please check if the code works: I can't.) Change html-tbl-writer-function in the file include/layout.scm to something else but 'html3-write-tbl or use -R HTML2 at the command line.

```
<figure id=tblexam>
<table<tbl>
<pattern<left<center<sep double<decimal align=",">
<r><bf/Name/ | <bf/Group/ | <bf/V/
<sep>
<r> Fred | M | 3,4
<r> Sian | F | 5,78
<r> Tiger| F | 100
<pattern/<right<center<right>/
<sep double>
<r>A | &lt;=> | B
</tbl<table
<caption/Example table/
</figure>
```

Will give you what figure 2.1 shows.

Questions:

- Better have the freedom to mix the patterns with the rows (as it is now) or to group all patterns before all the rows.
- Is the short reference character "|" a good choise for the column separator?
- How much requests will come for row spawning cells? (They eat more steam in implementation, thus it will take some more time.)

2.5.14.2 Next

How about a HTML-3 like syntax. This is even harder to implement on the existing targets (while it's obviously easier on viewers – but these are internals). Open questions:

- I, personaly, dislike the very verbose syntax of HTML-3 tables. How is yours? Better have <r> and <c> tags (and "|" as short reference for the latter) to separate row and columns or the long syntax of HTML-3. The latter will mess up your source code, but has the advantage to be the same as within HTML. (At the other hand why should we be compatible with the stuff we compile out of the source?)

If the above mentioned ways to create tables don't suit your needs you can use notations see 2.17. And you might consider utilizing the <inline> element (see 2.5.13) to write the table using other notations as available local at your site (see 2.18).

2.5.15. Equations

There is a limited support for equations at the moment. As they are used they'll be added. To give an example what's

Name	Group	V
Fred	M	3,4
Sian	F	5,78
Tiger	F	100
A	<=>	B

Figure 2.1. Example table

supported at the moment:

```
<quote>

<bf/Lemma:/ <math/&alpha; &isin; <set/M<sup/200/<sub/4// &cap;<set/N//
</quote>
```

Will yield:

Lemma: $\alpha \in \underline{M^{200}{}_4} \cap \underline{N}$

Future version will probably support the same features as HTML-3. (As this is in fact simply a SGMLish translation of LaTeX's idea of equations and the latter is commonly treated to be the best way to write equations.) The question remains: is this the best way or does anybody have a better idea?

As mentioned the support on equations is limited at the moment and intented for simple formulas. The translation into formating instructions is straight forward implemented and not always reliable. If you intent to use LaTeX as backend you want to use `<inline latex>` (see. 2.5.13)and native LaTeX coding instead.

It's not a big task to extend the formating rules to filter things like that through the PostScript backend and convert it into say GIF for HTML. But nobody came around to program it yet.

2.5.16. Foreign Languages

sdc will insert standard text phrases if appropriated. These depend on the language of the document.

In the simple case you can choose the language for the whole document. This is done by the document declaration. That is:

```
<!doctype document public "-//JFW//DTD Document//DE" >
```

will produce a German document and:

```
<!doctype document public "-//JFW//DTD Document//EN" >
```

an English one.

Furthermore the tags `report, document, chapt, sect, sect1, sect2, lang` have an attribute `lang` which will temporary change the language in the enclosed part.

2.6. Creating Indexes

sdc creates an index for a document if at least one `<index>` tag was used in the text.

An index tag can appear at every place where text is allowed except for headlines, which are terminated by.

For the index tag the following attributes are defined:

`id` The topic which will appear in the index section. This attribute is required.

`sub` An optional subtopic.

Creating useful indexes is an art by itself. Therefore you should choose the attribute values carfully. The most common (recomended) way to use indexes is like that:

??

```
<sect id=Indexes>Creating Indexes
<index id=Element sub=index
<index id=Indexes>
```

Please *note*: Because of an unresolved problem it's better to use the short notation as above. That is: *don't* close the `<index` tags (omit the >). Otherwise the PostScript output will create an empty paragraph.

As mentioned, the `<index>` tag is not restricted to be used immediately after the section start.

2.7. Bibliography database

sdc can use one or more databases for bibliography. If items of them are referenced (by a `<ref>` see 2.5.5), a section is automatically appended to the document and the referenced items are listed.

To use a data base you need to include the database file for instance like this:

```
<!doctype report public "-//JFW//DTD Report//DE" [
<!entity bib system "intro-bib.sgml" subdoc>
]>
```

And you *have to* reference the entity later on anywhere in the document like this: `&bib;`. But be sure to do so at some point where data is allowed, e.g., where the first paragraph begins. (Otherwise you'll violate SGML rules.)

The database has the following structure (with repeated BIBL's):

```
<!doctype bibdata public "-//JFW//DTD Bibliography//EN" [
]>
<BIBL
     Tag="SGMLGuide"
     Author="Martin Bryan"
     Title="SGML an authors guide"
     Publ="Addison Wesley Publishing Company"
     Year="1993"
     exc= "not avail"
     ISBN="...."
     COM="a comment on the book"
>
```

Don't write data in this file, only start tags as the one above.

This data base will be extended (some day) to support at least all features a `bibtex` data base supports.

Please note: If you get warnings about items not in the data base, then the numbers within the references may be wrong. Those numbers are only correct if all references could be resolved.

2.8. Conditional Inclusion

You can always include text depending on the definition of some parameter entities as you can from `sgmls`. That is you write a marked section like this:

```
<![ %Name [
conditional included Text
]]>
```

Where name is a prior defined parameter entity. Refer to section 2.14 for predefined entities and conventions with sdc. To exclude the text by default you need to have a entity definition like the following in the document type definition:

```
<!ENTITY % Name "IGNORE">
```

Now you're able to include the marked section by a command line switch `-i Name`. This will pretend, that a entity definition

```
<!ENTITY % Name "INCLUDE">
```

is seen from the parser, which will overwrite the other one.

??

Depending on the target format (the -O switch) and the public text language some parameter entities are predefined. See 2.14 for details.

2.9. Slides

Slides are extracted from ordinary documents, better say only from documents of the types `document, report, book` and `Lektion`.

Everything enclosed by a `<slide>` tag goes onto one slide. (Or more than one slide, we can't control what happens if a slide gets overfilled.) Everything outside of those tags is simply dropped.

Slide tags might appear wherever a `<p>` tag is allowed.

If you have more stuff in the document as you want to have on the slide (as it is usually the case) you can exclude parts by marked sections, see 2.8.

Each slide might have an optional first `heading` element. It's contents goes into the title of the slide and is simply dropped for all representations but slides. If the `heading` element is left out, the title used for the slide is that given to the last division (like chapt, sect sect1) seen. The slides are grouped by the sect's of the document. So if all you want is a document full of slides, just write one and put `<slide>`'s in instead of paragraphs.

As there is only a "Slide" entity defined if you generate your slides but the usual case is to exclude stuff from them put three lines like:

```
<!ENTITY % Slide "IGNORE">
<[ %Slide [ <!ENTITY % noSlide "IGNORE"> ]]>;
<!ENTITY % noSlide "INCLUDE" >
```

This way noSlide is defined if Slide is not. Why? SGML say's the first definition wins. If you define Slide via the target noSlide is defined to ignore otherwise to include.

2.10. Personal Data

At some points `sdc` will insert person dependant text like the default for the authors name and institution.

These are defined in a document called `personal.data`. Usually it's to be found in the directory `DOCPATH` points to (see 2.3).

There is a file called `example-personal.data`in the standard library directory to be copied and adapted.

The following general entities *must* be defined from this file. (They are used to be the default for the corresponding attributes)

myself The name of the author.

my-Inst The institution / organisation.

2.11. Appendices

Appendices are all the sections which follow the `<appendix>`-tag.

2.12. Large documents

Large documents can be spread over individual files. These files together form the document. In principle there are two way to declare them.

1 As a General Entity (see 2.16).

2 As a SUBDOC General Entity

The first form will insert the whole document instead of the entity reference. This means, that the markup is part of the document, the entity is included in. This restricts what you can put into the document. but is has the advantage, that cross references work from outside into and from inside out.

Example: within the `<doctype ...` one can have a entity declaration like:

```
<doctype ... [
```

```
...
<!ENTITY t.desc system "descript.text" >
...
]>
```

and later on, at the position of the document where the contents of the file is to appear a usual entity reference like `<t.desc;`. (Note that you can not cross reference to that piece of the document as a whole, but to the id's defined within.)

sdc won't do anything interesting with the first form. It's already handled by the SGML parser.

The second form, subdoc entities, makes the part to be included a document of it's own. That is, it can (could) be used without being included. Hence you can't put references from inside out. SGML also disallows references from outside into it.

Those subdoc entities are restructured to form a division at the place the references occur, e.g., if you are within a section and reference a subdoc entity you get a subsection holding the text. sdc achieves this by preprocessing the subdoc entity in a smart way. It re-tags the dcoument as if it was taged according to the document-DTD.

As explained you can't do any cross referencing between the included document and the outside document. But you can reference to the subdocument as a whole by using it's entity name.

Example: the entity declaration changes from the above form to something like:

```
<doctype ... [
...
<!ENTITY bib SYSTEM "intro-bib.sgml" SUBDOC>
...
]>
```

Note that the declaration is extended by the `SUBDOC` keyword. And remember that documents included this way must start with a `<doctype...` declaration while those suitable for the first form must not.

Those subdoc entities are included the same way, that is in the example by an entity reference to `lt;bib;`. In this case the entity name (bib) becomes available for cross referencing (you can do `lt;ref id=bib//` in this example).

> You can't put those references everywhere you might want! Read ahead.

If you have such a reference to an entity which contains a subdocument at *the end* of the text of a division that's fine (but not recomented, see below). This way it will become a subdivision of the current one. But if there was already a subdivision open (e.q., a `<sect>` was there) it would become a subdivision of this division. If you want it at the same level as the last, the obious solution is to put a corresponding end tag first (in the example a `</sect>`). This is *not* enough! When we are "between" the divisions, there is no data allowed! Hence the reference is rejected by the SGML-parser. You need to wrap the reference with a `<InclDiv>` tag.

This is sort of a backdoor. It allows you to enter data and therefor the entity reference. "Real" data within a `<InclDiv>` tag is silently skipped, but subdocuments are processed correct. It's possible, but again not recommented, to have more than one reference within an `<InclDiv>` tag. The recommented way to reference `SUBDOC` entities is:

> Write those SUBDOC entity references like
>
> ```
> <InclDiv>&1st-ent;
> <InclDiv>&2nd-ent;
> ```
>
> At the and of a devision (like it's subdivision) to make them subdivisions of the latter. Or write
>
> ```
> ... The end of the last section.
> </sect1>
> <InclDiv>&subdoc;
> ```
>
> To make it a division at the same level as the one just closed (with the `</sect1>` in this example).

??

Please note that the tag name is written in mixed case. Actualy that's not nessesary. Tags can be in any case (by default). But if someone changed the SGML-declaration to use case on tag names all tags would have to be typed in lower case except this "InclDiv" which would be in mixed case as here.

The mixed case has been choosed because I think that the name is not too intuitive (it's a run together of "include" and "division"; I want to reserve the name "include" for something more intelligent) and I consider it special enough to deserve the attention of the reader of the document source.

2.13. Literate Programing

sdc supports so called "literate programing". That is the source code and it's documentation are mixed. To get the plain code it is first to be striped from the documentation.

With sdc it is possible to have the sources which goes into one file spread over a set of files, i.e., one document and also to have one document contain the source of a set of files.

To do literate programing the code is surounded by the `<literate` tag. The literate tag has one attribute called `file`, the name of the file where the code goes. This file is opened at the first occurence and closed at the end of the work of sdc. If no file is given the last used is implied if none at all has been specified prior standard output is used.

Within the `literate` taged area there may be anything what can appear in normal text. But no formating is applied to the output if target literate is choosen. The typical (intented) contents of it are verb and rverb elements and may be plain text paragraphs. Because of the formating applied on other target formats plain text paragraphs are only a good idea if the code (or comment) is readable after reformating.

With the features of replacement and conditional inclusion this form of literate programing might be useful for simple preprocessing.

2.14. Parameter Entities

SGML knows two kinds of entities: general entities and parameter entities. While general entities can be referenced anywhere, parameter entities are restricted to the meta level where the document structure is defined. This is at most only the document type description. But also marked sections can use parameter entities for conditional inclusion (see 2.8).

To avoid conflicts between user defined parameter entities and predefined parameter entities there is one convention: All parameter entities defined by sdc begin with a upper case letter. User defined parameter entities should therefore begin with a lower case letter. This is by no means enforced but a recommented convention.

Depending on the target format the following parameter entities are predefined. Only the one associated with the target format is defined to `INCLUDE` all other yield `IGNORE`.

LaTeX for LaTeX processed documents.

Lout for PostScript output through Lout

HTML for HTML pages

ASCII for ASCII output

Info for output in Info format

Literat for literate programing

RTF for RTF output

Slide together with Lout for extracting slides.

Depending on the public text language parameter entities named as the ISO 639 defined language codes will be define to `INCLUDE`. The other again to `IGNORE`.

Currently only the definition to include is handled. It's up to you to define them to `IGNORE` in you doctype diretive. Also only English and German are supported at the moment.

Here the codes reserved for this purpose:

EN: English, DE: German, FR: French, GR: Greek, IT: Italian, NL: Dutch, ES: Spanish, PT: Portuguese, AR: Arabic, HE: Hebrew, RU: Russian, CH: Chinese, JA: Japanese, HI: Hindi, UR: Urdu, SA: Sanscrit.

2.15. Changing the Layout

As it is not intended to have particular formating instructions or something like that within the document, it can't be done.

To change the formating use the `-R` option to load a scheme file. If you do so, be sure what you do! You might want to have a look at the file `include/layout.scm` as the definitions there are most likely to be changed due to taste. Here an example of such an file showing how to change the initial font for lout.

```
(doc-preprocess-hook
 'add
 (lambda ()
   (if (equal? doc-output "lout")
           (eval (set! lout-initial-font "Times Base 12p")))))
```

Having this within a file, say `TB12.scm` call sdc like

```
sdc -R TB12.scm -Ops -o output.ps text.sgml
```

2.16. General Entities

Currently there are general entities for setting mathematical symbols predefined. Their names are those defined by Addison Wesley.

Also the set of general entities for Greek characters are defined as by Addison Wesley.

To see these set of entities and their repesentation, format a document like the following into the target format you need.

```
<!doctype document public "-//JFW//DTD Document//EN" [
<!ENTITY f.AWm SYSTEM "AWmaths.text" >
<!ENTITY f.AWg SYSTEM "AWgreek.text" >
]>
<document face="2c 2s nidx">General Entities

<sect>AWmaths

&f.AWm

<sect>AWgreek

&f.AWg
```

Prior version had a problem with the < Entity. There where two flavors, < and <. The problem is solved, see the changes section.

2.17. Notations

SGML provides so called notations to define entities which use "foreign" descriptions of their content. That is, the entity might be a file containg figure drawn with say tgif. The interpretion of system notation is implementation dependant.

You might define notations. And some entities using them. sdc provides an interpretion for notations which are declared to be `SYSTEM`.

The system identifier is used to start external commands. Prior the command execution the following macros are expanded:

`%s` the system identifier of the entity

`%S` the flatend system identifier of the entity

`%f` the complete file name of the entity

`%F` the flatend file name of the entity

`%%` a single `%`

“Flatend” means that characters special to the file system are converted into characters elegible for regualr file names. Per default ’/’ are converted into ’_’.

The characters to be converted are set using `file-system-set-meta-characters`.

If the notation is to be applied to inlined code, a temporary file is used.

Example:

```
<!DOCTYPE document public "-//JFW//DTD Document//DE"[
<!NOTATION cat SYSTEM "cat %f" >
<!ENTITY f1 SYSTEM "file1" NDATA cat>
<!ENTITY f2 SYSTEM "something" NDATA cat>
]>

<document
face="2s 2c"
>Notation Demo

Some text goes here.
&f1;

And now call &f2;.
```

For an extensive example see `doc/nottest.sgml` in the sdc distibution. This shows the “real world” notations provided by default by sdc.

The above example declares a notations named “cat” which starts the external program `cat` with one argument, the full filename of the entity the notation is apllied to.

Then two entities (f1 and f2) are declared. These use both the notation `cat`. Their system identifiers are “file1” and “something” (i.e., their content is stored in these files). And their content is declared to be in the notation `cat`. (That is done by the `NDATA` like notation data keyword followed by the name of the notation i.e., `cat` for the example).

When the entities are referenced (`&f1;` and `&f2;`) the command line associated with the notation (i.e., `cat %f`) is executed (after the expansion of macros as explained above). The output of this command is feed into the backend/output of sdc.

Make sure, that the output generated by the external programm is suitable for the target format. If needed, use conditional definitions for the notations depending on the target format. (See section 2.8 for target dependant defined parameter entities.) Needless to say, the gain in comfort and features is paid by a loss in document portability.

A possible application could be to have a picture stored in some format (like that of xfig). If there is a program `fig2dev` which translates this format into encapsulated PostScript you may wrap it by a shell script like this:

```
#!/bin/sh
SRC=`basename $1`
TARGET=`basename $1 .fig`.eps
fig2dev -L ps $SRC > $TARGET
echo "@IncludeGraphic{" \"$TARGET\" "}"
```

If you use this notation with the Lout backend to produce PostScript, this will insert the picture on every reference to the corresponding entity.

2.18. Local Features

The following notations are defined by default:

`ignore` which will just print a note, that it has been used. It's intended to get rid of the functionality implied by others.

`eps` This is Encapsulated PostScript.

`fig` This is to be used with figures drawn by `xfig`. It will automatically retrieve the referenced file and transform it into the proper format for the target.

`lfig` Figures drawn using the @Fig package of Lout. See the Lout user manual[2] for a description.

`roff` Preprocesses `tbl` and other roff code. Be sure not to exceed one page with the image you get from this notation.

`latex` Pieces (especially supposed to be formulas) ofLaTeX code.

`tgif` This is for figures drawn by `tgif`.

To use them you need to extend the head of the document like:

```
<!doctype document public "-//JFW//DTD Document//DE"[
<!ENTITY fig1 SYSTEM "figure1.fig" NDATA fig >
<!ENTITY tbl1 SYSTEM "table1.eps" NDATA eps >
]>
```

Then you can use it as described in section 2.5.4.

2.19. Installation

See the file INSTALL.

2.20. Changes

- (22nd Okt 96) search `sgmls` as `sdc-sgmls` in the `bin` subdirectories of the lib dir.
- (21st Sep 96) introduced path name flatening to protect NDATA entities stored in different subdirs of the DOCPATH from each other. The "/"'s get translated into "_"'s now. (see files.scm about meta characters.)
- (7th Juli 96) Version 1.0 cleaned the Hack in the DTD's to include subdoc entities. Now they should be wrapped with an `<InclDiv>`.
- (26th June 96) arguments to rc files
- (19th June 96) Version 0.7: Introduced reparsing of the document. Implementation based on streams and an recursive descendant parser.

  Because the old code is left in, some files are hardly clean and easy to read.
- `SUBDOC` entities supported. You can include complete documents within others and get them restructured as divisions of the outer document.
- DTD's changed to use mixed content model. Not explicit <p> tags nessesary anymore.
- `<newpage` supported within list items.
- Tgif supported as notation.
- Most (the computational intensive) notation handling based on Makefiles.
- `<quote>` element has got an `style` attribute.
- `-D` option added. It adds a directory at front of the path given by the environment variable `DOCPATH`.
- The `&ltc;` is not needed anymore. (But still allowed)
- (25th. Oct. 95)`SGML_PATH` is no longer used (**but overwritten!**). The new name for the path pointing to the document tree is `DOCPATH`. It is a simple colon separated path, pointing to the directories to be searched for document entities.
- (25 Oct 95) Proposed tables work a little with Lout (they work as long as there are no row/column spawning cells). Unforunatly the implementation revealed limits of Lout. As there is obviously no way to implement the full semantic without changes to Lout, and the latter are not within reach.

??

- rc files added
- library is now a path

2.21. Problems

If you get any problems don’t hesitate to mail me:

`<joerg.wittenberger@inf.tu-dresden.de>`

Known Problems:

- Ghostscript version has a problem 3.3 writing ppm files. If sdc refuses to generate gif files try ghostscript version 2.6.1.
- LaTeX output dischards leading whithespaces in <verb> and <rverb> elements. I don’t know how to coerce LaTeX to keep them. There is a macro \psedoverb in the converter file for LaTeX if you have an idea how to correct this macro please drop me a note.
- Quote characters are internal translated into markup. They can’t spawn paragraph boundaries.

Chapter 3. Target formats of typeset

Typeset supports various "target formats" i.e., backends. All these formats have some special properties, are supported to different extent and are partially custumizable. Here details are discussed.

3.1. PostSript

The PostSript output is generated by a special PostScript formater. Typeset generates only the input for this formater and invokes it for the PostSript output.

There is support to choose the backend formater in pinciple, but currently only Lout and LaTeX is supported. Future versions will at least support LaTeX + dvips.

Why is Lout the default? Just because it's the one simple enough that I learned a bit how to program it. And it supports most of the features someone needs to produce neat documents. Unfortunatly it has some drawbacks discussed in the Lout section3.3. I must also admit that LaTeX output looks a little better, but that is a matter of taste. See LaTeX section3.4 how to use it output.

3.2. ASCII

For the ASCII target applies the same as for PostScript. The default formater is also Lout. Maybe some day there will be support for roff. See Lout section3.3 for discussion.

3.3. Lout

Lout is the default PostScript backend of typeset.

Even if not described in the standard values for the `-O` Option also `lout` is a valid parameter to the `-O` option. It's not described among the standard values because it's not considered to be a final target.

The most important features of Lout are it's easy to learn programing language, it's capability to denote equations, pictures, graphs and even "special effects" (like rotated text) without much hassle and the fact that it's relativly small on the disk.

All the document types which come with typeset have a representation in the Lout target. For the `brief` it's the only target defined. See 3.3.3 about the manpage document type.

3.3.1. The `face` attribute

The face attribute is a string of tokens delimited by spaces.

The following values of the `face` attribute are recognized:

1S 2S These change between one side and double side printing.

1C 2C These switch between one column and two column for the main text. Some text chunks are one column in any case (e.g.,the table of contents, abstract, intro, preface) index text is two column in any case.

nidx This supresses the generation of an index even though `<index>` tags are used in the document.

By default simple documents use `face="1S 1C nidx"`, report use `face="1S 2C"` and books `face="2S 1C"`.

3.3.2. Notations with Lout

For all notations suitable to be used in conjunction with the Lout target there must be a way to convert the code either into Lout code or into Encapsulated PostScript.

Typeset as it comes has these conversion defined for a couple of notations (e.g., notation `fig` for figures drawn with `xfig`).

Notations which are converted into Encapsulated PostScript can't really determine if the `.eps` file is no longer nessary. Therefore it is not deleted by typeset.

It is always a good idea to move to an empty directory to invoke typeset and supply the full file name to the source. This way you will end up with your document and you can see with ease which files are no longer needed.

3.3.3. Problems

But the old Lout had also its drawbacks. Unfortunalty typeset reveals bugs and after all missfeatures all the time. Those keep me for instance from a better table implementation.

The old version of Lout were unable to break indented junkt of text over columns and pages. This is solved by the new release, but typeset reveald a bug in that code. Thus there is a chance left to get bitten by this bug. If it appears, long junks will not be broken. Moreover the spacing between the paragraphs behaves very funny in this case. It is likly to happend only when you have nestet <desc> elements and more than one paragraph in the inner description list. If this appears to you please send me a test file.

An finally there is the problem with Lout, that it may fail sometimes to resolve cross references on heavy loaded machines or if disk space is "tight". (Where tight could be a couple of Megabytes.) This should be circumvented now because most cross referencing is done without the lout (only page references are left to lout). But I can't test it.

3.3.4. Custumization

Lout has its own way for extensive custumization. This includes all sort of things like hyphenation exceptions, definitions of new symbols, page background and so on. This is not the place to discuss them. See the documentation of Lout for details.

Typeset supports the widely nessesary custumizations. To change these options go to the file `layout.scm` in the `include` directory of the installation. The following options are custumizable:

page size Only the named page types of Lout are supported. These are: Letter, Tabloid Ledger, Legal, Statment, Executive, A3, A4, A5, B4, B5, Folio, Quarto and 10x14. Change the definition of `lout-pagetype` to the size you want.

page margins The left and right margins of the document are set from typeset due to the request of single or double page printing.

For single page printing both margins are set to the same value `lout-both-margin`, for double page printing the left side of odd and the right side of even pages are the same `lout-inner-margin` and the other are set to `lout-outer-margin`.

initial break The initial break used by typeset is also open to custumization. But it's very unlikly that someone need to change it.

Future versions will open the formating of paragraphs to custumization.

3.4. LaTeX

You can produce LaTeX files from typeset. To produce DVI file you need to run LaTeX on the result by hand. For PostScript output through LaTeX you also need to run dvips.

Currently only the DTD's book, report and document are supported for LaTeX.

The file is usually ready to be feed to LaTeX. There is no need for `bibtex` or `makeindex`. Eventually you might adjust the substyles used. See the custumization section3.4.4 how to prevent this.

3.4.1. The `face` attribute

The same values as for Lout are supported see 3.3.1 for description.

3.4.2. Notations with LaTeX

The following notations are supported with LaTeX:

eps encapsulated postscript. Note: a temporary eps file is copied into the current directory and left there

fig xfig drawn pictures

latex Code unconditional passed to the backend

lout plain `lout` code

lfig lout code using the @Fig package of Lout

roff roff code which will be converted into eps by `groff`.

tgif tgif drawn pictures

As for some other target formats it is a good idea to format the document in a empty directory allowing you to delete files left over with ease.

3.4.3. Problems

LaTeX is huge and most installations differ. The package aims to run at most plain installations. Only the inclusion of EPS figures depends on either the substyle `epsfig` or the `graphics`package.

Which one is used has to be defined in the installation process.

Also to be defined in the installation process is whether LaTeX version 2.09 or LaTeX2e code is generated.

Typeset aims to be a "don't worry application" for the author. Regarding LaTeX this implies that some characters are set in a save way regardless whether or not this is nessesary in a certain case. E.g., because the characters < and > yield strange results in some environments they are set using `{\tt >}` respectivly `{\tt >}` except within `<math>` elements.

Also LaTeX offers an extensive set of symbols to write math formulas. Unfortunatly it is quite hard to generate them propper for what typeset already offers. So better use typeset's way only for simple formulars and <inline latex> for complicated formulas wich are written in native LaTeX notation then.

I've also got a report about a version which discard the leading (or all?) blanks in <verb> elements.

3.4.4. Custumization

To custumize the LaTeX output you need to change the file `include/layout.scm`. The following variables are available:

latex-latex-type Switch between LaTeX 2.09 or LaTeX2e style.

latex-styleoptions This is a list of strings. By default only `"epsfig"` is included. For LaTeX 2.09 this seems to be a good choise, for LaTeX2E it's probably the empty list.

latex-packages A list of packages to be included by `\usepackage` into the LaTeX source. By default only `[dvips]{graphics}` is in this list.

latex-preamble This is a list of strings which are put into the preamble of the document just before the `\begin{document}`. By default `latex-a4-preamble` is included into this list.

For german documents the substyle `german` will be included into the list of substyles. For further custumization (e.g., other languages) you'll need to go into the code of `target/latex/preparse.scm`. If you do please drop me a note so I can include the support.

3.5. HTML

3.5.1. The `face` attribute

Only one token is used from the face attribute: `nidx`. If it is present the generation of an index is supressed.

3.5.2. Splitting

Except for the book document type one document becomes one HTML file (defined by the `-o` option to typeset). For the document type `book` the document is split at the chapter boundaries into single files. For the names of these files the base name of name given to the `-o` option is extended by *-number*. E.g., given a command line of:

```
typeset -O html -o doc doc.sgml
```

And assumed the file doc.sgml contains three chapters, you will end up with (at least) 6 files named `doc.html, doc-1.html, doc-2.html ... doc-6.html`. These contain the top level stuff in `doc.html`, the single chapters – one per file – and also the index and references get their own file.

If entities of other notations are used (e.g., figures) you will get some more files following the same naming scheme (with different extensions – most likely ".gif"). Refer to 3.5.3.

3.5.3. Notations with HTML

For the HTML target, notations are best converted into GIF files. To do this conversion typeset will by default invoke ghostscript and the pbm-tools for Encapsulated PostScript. The files are named after one of two schemes. For external entities which already have a filename, the basename of it is used (with extension ".gif"). If the notation is used inline, a new "subfile" is created with the basename of the target file (given to the `-o` option) appended with a dash and a running number. The extension is also ".gif". See the splitting section3.5.2 for details.

3.5.4. Problem

Because of the possible splitting the filenames of the file(s) produced are compiled into the output. Therefor you can't rename them after compilation anymore. You need to be aware of this and give a filename *without* any leading directory component to the `-o`. option.

3.5.5. Custumization

There is not too much to be custumized for the HTML target. Only for manpages: the file `manpage.scm` in the `include` directory is to be changed. The procedure `html-make-man-ref` receives a string (the id given to a `<ref t=m id=`*string*`//` tag) and has to return a list of string. The concatenation of these strings must form valid HTML code (either a URL to some server providing man pages or text to be included in the document.

3.6. Info

Only the document types `document, report` and `book` have a representation in the info format. Due to the limitation to plain text some, tags like character formatings are ignored.

The `face` attribute is completly ignored.

For each division of text (i.e., the whole document/ report/ book, chapters, section and subsections) a node, is generated with a menu containing each division of the next level.

Other than the preceding version, the Info output is no longer spread into defferent files. Todays computers are usually powerful enough not to need this, and it would blow up the code too much.

Only notations which have a plain text representation can be used with the info target.

3.7. man

For the manpage target only the document type `manpage` is really supported. The intention of this target is not to print everything from the man command but to produce pages suitable to be stored in the online manual of unix systems.

To produce a printed version of manpages you should use the manpage produced by typeset and feed it into the `nroff` command of your system.

Only the notation `roff` is supported with this target.

Custumization

The file `manpage.scm` in the `inlcude` hold a translation table from the symbolic short name of the section into the numbers and the long names. Some systems use a different order (numbering). Adapt it to reflect your system.

3.8. Literate Programing

For literate programing the filenames written are part of the program hence determined within the document. Only if no file name is given for a part of literate program it's written to the standard output.

There is nothing to be custumized for this target.

3.9. RTF

The RTF target is the least supported one. Only the document types `document` and `report` are supported by now and even these not to the full extend.

The RTF target is intented to open a way to use parts of a document with the widespread MS tools. Unfortunatly it's pretty hard to support this format. Even page size and font information is stored in and to make it worse the "reference application" (guess which) treats it different than the definition published by Microsoft.

But for the restritions Wordprocessor carry, RTF formated text is not supposed to look as professional as others anyway.

3.10. Slide

The slide target works simillar to the literate target. Only the parts of the source enclosed by the `<slide>` tag are extracted. These are formatted using the lout backend with it's overhead transparencies support.

Appendix A. Greek characters

Just to see what's about the ref's 1 and 2.

α α

&Agr; Α

β β

&Bgr; Β

Γ Γ

γ γ

δ δ

Δ Δ

ε ε

ϵ ε

&Egr; Ε

ζ ζ

&Zgr; Ζ

η η

&EEgr; Η

θ θ

&thatav; ϑ

Θ Θ

ι ι

&Igr; Ι

κ κ

&Kgr; Κ

λ λ

Λ Λ

μ μ

&Mgr; Μ

ν ν

&Ngr; Ν

ξ ξ

Ξ Ξ

&ogr; ο

&Ogr; Ο

π π

Π Π

ρ ρ

&Rgr; Ρ

σ ς

ς σ

Σ Σ

τ τ

&Tgr; Τ

υ υ

ϒ Υ

φ φ

ϕ ϕ

Φ Π

χ χ

&KHgr; Χ

ψ ψ

Ψ Ψ

ω ω

Ω Ω

Appendix B. Math characters

− −

+ +

· ·

× ×

***** ∗

÷ ÷

= =

◊ ◊

< <

> >

± ±

∓ ∓

⊕ ⊕

⊖ ⊖

⊗ ⊗

⊘ ⊘

⊙ ⊙

◯ ◯

○ ○

• •

≈ ≍

≡ ≡

⊆ ⊆

⊇ ⊆

≤ ≤

≥ ≥

⪯ ⪯

⪰ ⪰

∼ ∼

≈ ≈

⊂ ⊂

⊃ ⊂

≪ <<

≫ >>

≺ ≺

≻ ≻

← ←

→ →

↑ ↑

↓ ↓

⇆ ↔

↗ ↗

↘ ↘

≃ ≃

⇐ ⇐

⇒ ⇒

⇑ ⇑

⇓ ⇓

&lrArr; ⇔

↖ ↖

↙ ↙

∝ ∝

&ssol; /

∞ ∞

∈ ∈

∋ ∉

▵ △

▿ ▽

/ /

∀ ∀

∃ ∃

¬ ¬

⊥ ⊥

ℵ ℵ

&mathsA; $\mathcal{A}$

&mathsB; $\mathcal{B}$

&mathsC; $\mathcal{C}$

&mathsD; $\mathcal{D}$

&mathsE; $\mathcal{E}$

&mathsF; $\mathcal{F}$

&mathsG; $\mathcal{G}$

&mathsH; $\mathcal{H}$

&mathsI; $\mathcal{I}$

&mathsJ; $\mathcal{J}$

&mathsK; $\mathcal{K}$

&mathsL; $\mathcal{L}$

&mathsM; $\mathcal{M}$

&mathsN; $\mathcal{N}$

&mathsO; $\mathcal{O}$

&mathsP; $\mathcal{P}$

&mathsQ; $\mathcal{Q}$

&mathsR; $\mathcal{R}$

&mathsS; $\mathcal{S}$

&mathsT; $\mathcal{T}$

&mathsU; $\mathcal{U}$

&mathsV; $\mathcal{V}$

&mathsW; $\mathcal{W}$

&mathsX; $\mathcal{X}$

&mathsY; $\mathcal{Y}$

&mathsZ; $\mathcal{Z}$

∪ $\cup$

∩ $\cap$

∧ $\wedge$

∨ $\vee$

⊢ $\vdash$

⊣ $\dashv$

⌊ $\lfloor$

⌋ $\rfloor$

⌈ ⌈

⌉ ⌉

{ {

} }

⟨ 〈

⟩ 〉

| |

∥ ‖

**** \

√ √

∐ ∐

∏ ∏

∫ ∫

∮ |

∑ ∑

∇ ∇

⊔ ⊔

⊓ ⊓

⊑ ⊑

⊒ ⊒

§ ∐

&dag; †

&Dag; ‡

¶

&emptset; ∅ (not from AWmath)

Appendix C. Installation procedure

The installation is pretty straight forward, but if you have to install something else first, it may take some time.

1 Make sure you have sgmls installed somewhere in you path.

2 You need bigloo to compile the source (or change it to be useful for another scheme system). Thus probably better get bigloo.

3 If you want to use Lout for the backend install it too.

4 For EPS pictures in the LaTeX (prior to 2E) output you need `epsfig.sty`.

5 Then install the package itself.

Installation of the package:

During the installation the following rules apply:

- No file is created by the (basic) installation process outside of the base directory named DD (choosen in the toplevel Makefile).
- The runtime access to intalled files goes via the directory named SR choosen in the toplevel Makefile.
- After installation the directory where the distribution was untared may be removed.
- The documentation is installed from the DD directory and has it's own custumizable target directories.

The installation process:

Unfortunatly there is always a problem. Some systems don't have a full featured install other lack the -R switch on cp. There is a install program in the install subdirectory of the distribution. Use it if your's doesn't understand the -d switch. (This install also supports an environment variable INSTALLFILE, which can point to a file where a list of installed files is to be loged.)

1 Untar the distribution

2 Edit the Makefile. Usually all you have to set up is the base directory (SR) or only the "prefix".

   Set the variable `PARSEPROG` to either "sgmls" or "nsgmls". Do *not* use a full path here!

   Also decide whether to generate the LaTeX for the graphics package or the epsfig substyle. There is the variable `GRPHPKG` for. Set this to "grphcs" or to "epsfig". I don't know if epsfig works with LaTeX2e nor if graphics works with LaTeX 2.09. I recomented to use epsfig with version 2.09 and graphics with 2e.

   If you need, adapt the variables holding the make, cp and mkdir program as well as the mkdir options used through the installation process.

   There is still some support for the and the variable `REMLOC` left in. This is from the old version, may disappear some day and is no longer full supported. You can choose either local or remote (probably local). (The remote installation sets up a quite tricky environment, where some of the used commands are accessed on a remote host.)

3 **Only** for bigloo version 1.7:

   You need to change the source back to the old module language of bigloo.

   Within the module clauses (these is the first declaration within a file) of the files

   - `typeset.scm`
   - `files.scm`

   are comments like

```
;BEGIN BIGLOO1.x
; .....
;END BIGLOO1.x
```

Also further down within `typeset.scm` there is a block surounded with the same comments. It's a work around a bug within bigloo1.7.

Find these lines, remove the semicolons commenting out the lines between ";BEGIN BIGLOO1.7" and ";END BIGLOO1.7" respectivly comment out those line only for version 1.8.

4 Do "make all" to compile the executable and hardwire the installation directory in some runtime files (have a coffee).

5 Now the hand work part: got to include/layout.scm and set latex-latex-type either to 'LATEX209 or to 'LATEX2E. (The default setting is 'LATEX2E).

6 Do a make install. This will copy the distribution and the executable into the choosen DD directory and hardwire the SR name in some of the files.

7 Make a symlink from the executable (Named by `$(EXE)` in the toplevel Makefile, to be found in the `bin` directory; default name: sdc) to something anywhere in your path (like `/usr/local/bin/sdc`). (I don't recomment to move or copy the executable around cause updates and uninstall will be easier this way.)

8 Thats all you need, but it's better to set `DOCPATH` to the root of your personal document tree (see the typeset man page).

9 To install the documentation do a "make installdoc". This will install the documentation in the default places.

Because these places may be different in your system, you might want to cd into the doc directory of the distribution. Do a "make help" to see what you can do. Then edit the Makefile to reflect your system and do a "make doc" to install the documentation. If you have trouble with the local handling of the notations you'll get some warnings.

10 The `include` directory holds some files (i.e., `layout.scm`, `manpage.scm` and `language.scm`) which might to be custumized. See the comments and `doc/targets.sgml` or compile it into a format you like to read.

(e.g., sdc -O ps -o targets.ps targets.sgml)

11 There is a file `sgml/local.ent`. Change this to reflect your system, if you need. Things like localy defined notations go here. (Only for advanced users installation.)

12 If your use emacs you might want to extend `compilation- error-regexp-alist` with a line:

```
("SGML error at \\([, \t\n]+\\), line \\([0-9]+\\)" 1 2)
```

to get the goto-next-error funtions and the like in compile mode right. Uhm, for nsgmls you need a different pattern. But this should probably work out of the box.

I personally found the SGML-mode of emacs not suitable (these DTD's require to few markup to make this mode resonalble and the text formatting support was too limited for my taste). Therefor I (and others) hacked a slight modification. `bin/mode.el` holds the code for a new mode I like (especially for the folding properties). (Please note that this mode reflects personal taste.) It's far from perfect, especialy the highlighting is broken. Unfortunatly the 19.31 update broke the code. Users of older version might uncomment one line in the keymap to get the closing angle work again.

13 If you want to install automatic figure handling or some other notations on your system, see `doc/notations.sgml` (which is also the source for the some other files) to see whats going on. Among the packages involved there are: ghostscript, xfig, tgif, pnmtools, groff, latex and lout. But you are not forced to have them.

14 If you want install bbset, a program to correct/insert correct bounding boxes in EPS files, for use without sdc. It's installed as a part of sdc (in it's private bin directory) but might be useful for other things.

If you issue a "make instbbset" from the distribution directory you get it installed in `$(prefix)/bin` and the manpage in `$prefix/man/man1`.

15 See the file `doc/targets.text` for details about the formating in the various target formats. For easy reading format it into the type you like. E.g.,

```
sdc -O latex -o targets.tex doc/targets.sgml
```

Note, that you need to format the file `doc/target.sgmlwhich includes the other-` `doc/target.text`.

In the case you encounter any problems or find bugs, typos etc. please mail me: Joerg.Wittenberger@inf.tu-dresden.de.

Appendix D. Changes

- (22nd Okt 96) search `sgmls` as `sdc-sgmls` in the `bin` subdirectories of the lib dir.
- (21st Sep 96) introduced path name flatening to protect NDATA entities stored in different subdirs of the DOCPATH from each other. The “/”’s get translated into “_”’s now. (see files.scm about meta characters.)
- (7th Juli 96) Version 1.0 cleaned the Hack in the DTD’s to include subdoc entities. Now they should be wrapped with an `<InclDiv>`.
- (26th June 96) arguments to rc files
- (19th June 96) Version 0.7: Introduced reparsing of the document. Implementation based on streams and an recursive descendant parser.

  Because the old code is left in, some files are hardly clean and easy to read.
- `SUBDOC` entities supported. You can include complete documents within others and get them restructured as divisions of the outer document.
- DTD’s changed to use mixed content model. Not explicit <p> tags nessesary anymore.
- `<newpage` supported within list items.
- Tgif supported as notation.
- Most (the computational intensive) notation handling based on Makefiles.
- `<quote>` element has got an `style` attribute.
- `-D` option added. It adds a directory at front of the path given by the environment variable `DOCPATH`.
- The `&ltc;` is not needed anymore. (But still allowed)
- (25th. Oct. 95)`SGML_PATH` is no longer used (**but overwritten!**). The new name for the path pointing to the document tree is `DOCPATH`. It is a simple colon separated path, pointing to the directories to be searched for document entities.
- (25 Oct 95) Proposed tables work a little with Lout (they work as long as there are no row/column spawning cells). Unforunatly the implementation revealed limits of Lout. As there is obviously no way to implement the full semantic without changes to Lout, and the latter are not within reach.
- rc files added
- library is now a path

Appendix E. Bibliography

[1] *Martin Bryan*: SGML an authors guide; 1993 *Addison Wesley Publishing Company*

[2] *Jeffrey H. Kinston*: A User's Guide to the Lout Document Formatting System (Version 3); 1994 *Basser Department of Computer Science*

Index