

The xsbdoc Documentation Generator

Terrance Swift
based in part on code and documentation by
Manuel Hermenegildo and the CLIP Group
Version 0.1#1 (2001 / 11 / 1)

**An Automatic Documentation Generator for XSB Programs
Inspired by the Ciao lpdoc Document Generator**

This copyright needs to be worked out better before it can be released. Copyright 2002 Terrance Swift/XSB Group, Copyright © 1996-99 Manuel Hermenegildo/The CLIP Group. This document may be freely read, stored, reproduced, disseminated, translated or quoted by any means and on any medium provided the following conditions are met:

1. Every reader or user of this document acknowledges that is aware that no guarantee is given regarding its contents, on any account, and specifically concerning veracity, accuracy and fitness for any purpose.
2. No modification is made other than cosmetic, change of representation format, translation, correction of obvious syntactic errors, or as permitted by the clauses below.
3. Comments and other additions may be inserted, provided they clearly appear as such; translations or fragments must clearly refer to an original complete version, preferably one that is easily accessed whenever possible.
4. Translations, comments and other additions or modifications must be dated and their author(s) must be identifiable (possibly via an alias).
5. This licence is preserved and applies to the whole document with modifications and additions (except for brief quotes), independently of the representation format.
6. Any reference to the "official version", "original version" or "how to obtain original versions" of the document is preserved verbatim. Any copyright notice in the document is preserved verbatim. Also, the title and author(s) of the original document should be clearly mentioned as such.
7. In the case of translations, verbatim sentences mentioned in (6.) are preserved in the language of the original document accompanied by verbatim translations to the language of the translated document. All translations state clearly that the author is not responsible for the translated work. This license is included, at least in the language in which it is referenced in the original version.
8. Whatever the mode of storage, reproduction or dissemination, anyone able to access a digitized version of this document must be able to make a digitized copy in a format directly usable, and if possible editable, according to accepted, and publicly documented, public standards.
9. Redistributing this document to a third party requires simultaneous redistribution of this licence, without modification, and in particular without any further condition or restriction, expressed or implied, related or not to this redistribution. In particular, in case of inclusion in a database or collection, the owner or the manager of the database or the collection renounces any right related to this inclusion and concerning the possible uses of the document after extraction from the database or the collection, whether alone or in relation with other documents.

Any incompatibility of the above clauses with legal, contractual or judiciary decisions or constraints implies a corresponding limitation of reading, usage, or redistribution rights for this document, verbatim or modified.

Table of Contents

Summary	1
1 Introduction	2
1.1 Overview of this document	2
1.2 xsbdoc operation - source and target files	2
1.2.1 Documenting Libraries and/or Applications	3
1.3 Generating a manual	3
1.3.1 The Format File	3
1.3.2 Miscellaneous options	4
1.3.3 Graphical Formatting	6
1.4 Enhancing the documentation being generated	6
1.5 Other Usage Information	7
1.5.1 Separating the documentation from the source file	7
1.5.2 Writing comments to document version/patch changes	7
1.5.3 Cleaning up the documentation directory	8
1.5.4 Ensuring Compatibility with All Supported Target Formats	8
1.5.5 Generating auxiliary files (e.g., READMEs)	8
1.6 Troubleshooting	9
1.7 Known bugs and planned improvements (xsbdoc1)	9
1.8 Version/Change Log (xsbdoc1)	10
2 Enhancing Documentation with Machine-Readable Comments	11
2.1 Usage and interface (<code>comments</code>)	11
2.2 Documentation on exports (<code>comments</code>)	11
comment / 2 (pred)	
stringcommand / 1 (pred)15	
version_descriptor / 1 (pred)	3
21	
version_number / 1 (pred)	
time_struct / 1 (pred)22	
3 The XSB Assertion Library	23
3.1 Usage and interface (<code>assertions_props</code>)	23
3.2 Documentation on exports (<code>assertions_props</code>)	23
assertion_body / 1 (pred)	
assrt_status / 1 (pred)24	
References	25

Predicate Definition Index	26
Operator Definition Index	27
Concept Definition Index	28
Global Index	30

Summary

`xsbdoc` is an *automatic program documentation generator* for Tabled (C)LP systems written using XSB.

`xsbdoc` is an rewriting of the Ciao system's `lpdoc` to generate a reference manual automatically from one or more XSB source files. The target format of the documentation can be Postscript, HTML, PDF, or nicely formatted ASCII text. `xsbdoc` can be used to automatically generates a description of full applications, library modules, README files, etc. A fundamental advantage of using `xsbdoc` to document programs is that it is much easier to maintain a true correspondence between the program and its documentation, and to identify precisely to what version of the program a given printed manual corresponds. Naturally, the `xsbdoc` manual generated by `xsbdoc` itself.

Unlike `lpdoc`, `xsbdoc` does not use Makefiles, and instead maintains information about how to generate a document within Prolog *format files*. As a result, *xsbdoc* can in principle be run in any environment that supports the underlying software, such as XSB, `latex`, `dvips` and so on. To date, it has been tested on Linux and Windows/Cygwin.

The quality of the documentation generated can be greatly enhanced by including within the program text:

- *assertions* (indicating types, modes, etc. ...) for the predicates in the program, via the directive `pred/1`; and
- *machine-readable comments* (in the “literate programming” style).

The assertions and comments included in the source file need to be written using the XSB *assertion language*, which supports most of the features of Ciao's assertion language within a simple and (hopefully) intuitive syntax.

`xsbdoc` is distributed under the GNU general public license.

This documentation corresponds to version 0.1#1 (2001 / 11 / 1).

1 Introduction

`xsbdoc` is an *automatic program documentation generator* for XSB applications and modules. Based originally on the Ciao system's `lpdoc` [BibRef: ciao-man], `xsbdoc` generates a reference manual automatically from one or more XSB source files. The target format of the documentation can be Postscript, HTML, PDF, or nicely formatted ASCII text. `xsbdoc` can be used to automatically generate a description of full applications, library modules, README files, etc. A fundamental advantage of using `xsbdoc` to document programs is that it is much easier to maintain a true correspondence between the program and its documentation, and to identify precisely to what version of the program a given printed manual corresponds. Naturally, the `xsbdoc` manual generated by `xsbdoc` itself.

1.1 Overview of this document

This first part of the document provides basic explanations on how to generate a manual from a set of files that already contain assertions and comments. Examples are given using the files in the `doc` directory that generate this manual. These instructions assume that `xsbdoc` is installed as an XSB package.

Other parts of this document provide:

- Documentation on the formatting commands that can be embedded in *comments*.
- Documentation on the *assertions* that `xsbdoc` uses (those defined in the XSB `assertions` library).

All of the above have been generated automatically from the comments and assertions in the corresponding sources and can also be seen as examples of the use of `xsbdoc`.

1.2 xsbdoc operation - source and target files

The main input used by `xsbdoc` in order to generate a manual are Prolog source files. Basically, `xsbdoc` generates a file in the GNU `texinfo` format (with a `.texi` ending) for each Prolog file (see “The GNU Texinfo Documentation System” manual for more info on this format). Using these `texi` files, `xsbdoc` calls various applications to generate Postscript, PDF, HTML, or ASCII output.

The Prolog files must have a `.P` ending, although information is also taken from files ending in `.H` if the corresponding `.P` file is present. Depending on declarations from the format file used to generate the documentation, a given file may be documented as a library or as an application. In the first case, the `.texi` file generated for it will contain information on the interface (e.g., the predicates exported and imported by the file, etc.). In the second case, if it is an application, no no description of the interface will be generated

If needed, files written directly in `texinfo` can also be used as input files for `xsbdoc`. These files *must have a `.src` (instead of a `.texi`) ending*. This is needed to distinguish them from any automatically generated `.texi` files. Writing files directly in `texinfo` has the disadvantage that it may be difficult to adhere to all the conventions used by `xsbdoc`. For example, these files will be typically used as chapters and must be written as such. Also, the set of indices used must be the same as specified in the `xsbdoc` format file for the application. Finally, no bibliographic citations can be used in `.src` files. Because of

this, and because in the future `xsbdoc` may be able to generate documentation in formats other than `texinfo` directly (in which case these files would not be useful), writing files in `texinfo` directly is discouraged. This facility was added mainly to be able to reuse parts of manuals which were already written in `texinfo`. Note that if a stand-alone file needs to be written (i.e., a piece of documentation that is not associated to any `.P` file) it can always be written as a “dummy” `.P` file (i.e., one that is not used as code), but which contains machine readable comments).

1.2.1 Documenting Libraries and/or Applications

A manual can be generated either from a single source file (`.P` or `.src`) or from a set of source files. In the latter case, then one of these files should be chosen to be the *main file*, and the others will be the *component files*. The main file is the one that will provide the title, author, date, summary, etc. to the entire document. In principle, any set of source files can be documented, even if they contain no assertions or comments. However, the presence of assertions or comments will greatly improve the documentation (see Section 1.4 [Enhancing the documentation being generated], page 6).

If the manual is generated from a single main file (i.e., there are no component files) then a flat document containing no chapters will be generated. If the manual is generated from a main file and one or more components, then the document will contain chapters. The comments in the main file will be used to generate the introduction, while each of the component files will be used to generate a separate chapter. The contents of each chapter will be controlled by the contents of the corresponding component file.

1.3 Generating a manual

To use `xsbdoc`, the package must be first loaded via the command

```
[xsbdoc].
```

The top-level command to generate documentation is

```
?- xsbdoc(FormatFile,GenerationType).
```

Where `FormatFile` is a file specifying input files and formatting options to use, while `GenerationType` is specifies the graphical format to use: `postscript`, `html`, etc. We consider each of the arguments in turn.

1.3.1 The Format File

The `doc` library directory includes the *format file* `'xsbdoc_format.P'` used to generate this manual, and which can be used as a example of how to generate other manuals. Typically, the format file is put into a separate directory (say a subdirectory) from the code that it documents, and relative or absolute paths to the main file and the components are given so that they can be accessed. The main file to be used is indicated by the predicate `xsbdoc_main/1`, while component files are indicated by the predicate `xsbdoc_component/1` and are documented in the order in which they occur in the format file. For instance, in `'xsbdoc_format.P'`, the main file is indicated by the fact:

```
xsbdoc_main(' ../xsbdoc1.P').
```

while the two chapters are indicated by the facts

```

xsbdoc_component('comments.P').
xsbdoc_component('assertions_props.P').

```

`xsbdoc` uses as default directory search paths whatever directories have been asserted via the XSB predicate `library_directory/1` (See the XSB manual for more details).

Finally, if there are any citations in the manual, any bibtex files will have to be indicated through `xsbdoc_bibfile/1`. All the references will appear together in a *References* appendix at the end of the manual. If you are not using citations, then add the fact `xsbdoc_option('-norefs')` in the format file, which will prevent an empty 'References' appendix from appearing in the manual.

1.3.2 Miscellaneous options

Like `lpdoc`, `xsbdoc` can use default settings for many of the other options. However, if you wish to control the behavior of `xsbdoc`, you can do so through an array of options.

- Options that don't require parameters are indicated through `xsbdoc_option/1` facts. `xsbdoc_option/1` facts can be used to configure whether various pieces of information are included or not, along with many particulars of how the generated documentation looks. None of these options were used in generating this manual. The current set of values for `xsbdoc_option/1` are:

```
xsbdoc_option('-noauthors')
```

Omits all author names.

```
xsbdoc_option('-twosided')
```

Formats the documentation for printing on two-sided paper by starting each chapter and appendix on an odd-numbered page.

```
xsbdoc_option('-shorttoc')
```

Produces a shorter table of contents that does not include entries for definitions of individual predicates.

```
xsbdoc_option('-noverion')
```

Omits version information from various places in the manual — such as the title page and summary.

```
xsbdoc_option('-nosysmods')
```

Does not include system modules (defined as subdirectories of the root XSB directory) in list of libraries used for modules.

```
xsbdoc_option('-nochangelog')
```

Does not include a change log at the end of a chapter (cf. Section 1.8 [Version/Change Log (`xsbdoc1`)], page 10).

```
xsbdoc_option('-nopatches')
```

Does not include comments for patches within the change log (i.e. only major & minor revisions are included)

```
xsbdoc_option('-nobugs')
```

Does not include information on known bugs and planned improvements (cf. Section 1.7 [Known bugs and planned improvements (`xsbdoc1`)], page 9).

`xsbdoc_option('-norefs')`

Omits a *References* section regardless of whether citations are present.

`xsbdoc_option('-nopropsepln')`

Does not put each property in a separate line when `pred/1` assertions are commented for individual predicates (cf. the assertions for `comment/2` in Section 2.2 [Documentation on exports (comments)], page 11).

`xsbdoc_option('-nopropnames')`

Do not include property names when `pred/1` assertions are commented for individual predicates (cf. the assertions for `comment/2` in Section 2.2 [Documentation on exports (comments)], page 11).

`xsbdoc_option('-v')`

Generates verbose output and can be useful for debugging document specifications.

- Facts of the form `xsbdoc_index/1` determine the list of indices to be included at the end of the document. These can include indices for defined predicates, modules, concepts, etc.

`xsbdoc_index(pred)`

Index of defined predicates.

`xsbdoc_index(op)`

Index of defined operators.

`xsbdoc_index(concept)`

Index of cited and defined concepts.

`xsbdoc_index(global)`

Global index of cited and defined predicates, operators, libraries, applications, concepts, etc.

`xsbdoc_index(all)`

Includes all currently defined indices. *Limitations in the number of simultaneous indices* in some texinfo installations may occasionally prevent this option from working.

- The predicate `xsbdoc_startpage/1` changes the page number of the first page of the manual. This can be useful if the manual is to be included in a larger document or set of manuals. Typically, this should be an *odd* number. The default number is 1.
- `xsbdoc_papertype/1` allows selection between several paper sizes for the printable outputs (`dvi`, `ps`, etc.). The currently supported outputs (most of them inherited from `texinfo`) are:

`afourpaper`

The default, usable for printing on *A4 paper*. Rather busy, but saves trees.

`afourwide`

This one crams even more stuff than `afourpaper` on an A4 page. Useful for generating manuals in the least amount of space. Saves more trees.

`afourlatex`

This one is a little less compressed than `afourpaper`.

`smallbook`

Small pages, like in a handbook.

`letterpaper`

For printing on American *letter size paper*.

`afourthesis`

A *thesis-like style* (i.e., double spaced, wide margins etc.). Useful – for inserting `xsbdoc` output as appendices of a thesis or similar document. Does not save trees.

The default paper type is `letterpaper`.

1.3.3 Graphical Formatting

The second argument of `xsbdoc/2` indicates the target graphical format. Use of all options depends on having `tex` installed on your system.

The current options are:

- `dvi`, which generates a dvi format file, viewable by `xdvi`, `yap` and other viewers.
- `ps` which generates a viewable and printable postscript file. Use of this option depends on having `dvips` installed on your machine.
- `pdf` which generates a viewable and printable postscript file. Use of this option depends on having `dvips` and `ps2pdf` installed on your machine.
- `html` which generates html files in a subdirectory of the current working directory. The directory is named `<Main>_html` where `Main` is the base name of the file (i.e. the filename without a directory path or extension) specified by the fact `xsbdoc_main/1` in the format file. Use of this option depends on having `texi2html` installed on your machine.
- `ascii` which generates ascii files which are surprisingly nicely formatted to denote chapter and section headings, indices, and so on. Use of this option depends on having `makeinfo` installed on your machine.

Future versions will probably also include support for generating `rtf` files.

1.4 Enhancing the documentation being generated

The quality of the documentation generated can be greatly enhanced by including within the program text:

- **Import/Export Information.** `xsbdoc` uses information on exported and imported predicates in order to document the interface of a module. Because some users consider the XSB module system is awkward to use while developing code, `xsbdoc` also recognizes the directives
 - `document_export/1`, and
 - `document_import/1`

which are treated by `xsbdoc` exactly as `export/1` and `import/1`, but which are ignored by the XSB compiler.

In principle, only the (document_)exported predicates are documented, although any predicate can be included in the documentation by explicitly requesting it (see the documentation for the `comment/2` declaration).

- **Assertions** are directives that are included in the source program and provide the compiler with information regarding characteristics of the program. Typical assertions include standard compiler directives (such as `dynamic/1`, `op/3`, `use_variant_tabling/1...`), etc. However, the `pred/2` assertion can also be used to declare global, call, and success types and modes (see Chapter 3 [The XSB Assertion Library], page 23).

When documenting a module, `xsbdoc` will use the assertions associated with the module interface to construct a textual description of this interface. Judicious use of these assertions combines documentation of the program with improved debugging behavior. Improvement in debugging is possible because some assertions provide information on the intended meaning or behaviour of the program (i.e., the specification) which can be checked at compile-time (by a suitable preprocessor/static analyzer) and/or at run-time (via checks inserted by a preprocessor). (TLS: this portion is still under development for XSB).

- **Machine-readable comments** are also declarations included in the source program but which contain additional information intended to be read by humans (i.e., this is an instantiation of the *literate programming* style of Knuth [BibRef: knuth-lit]). In addition to predicate-level comments, typical comments include title, author(s), bugs, changelog, etc.

1.5 Other Usage Information

1.5.1 Separating the documentation from the source file

Sometimes it is convenient to include long introductory comments in a separate file from a source code file. This can be by using the `@include` command. For example, the declaration:

```
:- comment(module, "@include{Intro.xsbdoc}").
```

includes the contents of the file `Intro.xsbdoc` as the module description.

Alternatively, sometimes it may be convenient to generate the documentation from a completely different file. Assuming that the original module is `m1.P`, documentation can be kept in `m1_doc.P`, and included as a component in the format file. `xsbdoc` recognizes and treats such `_doc` files specially so that the name without the `_doc` part is used in various parts of the documentation, in the same way as if the documentation were placed in file `m1`.

Finally, there are several other mechanisms of including image files, predicate definitions and so on in machine readable comments, as is explained in Chapter 2 [Enhancing Documentation with Machine-Readable Comments], page 11.

1.5.2 Writing comments to document version/patch changes

Version comments (`:- comment(version(...), "...")`.) can provide useful documentation on how a given library or application has been changed or has evolved. Sometimes one would like to write version comments that are internal, i.e., not meant to appear

in the manual. This can, of course be done with standard Prolog comments (which `xsbdoc` will not read). An alternative and quite useful solution is to put such internal comments in *patch* changes (e.g., 1.1#2 to 1.1#3), and put the more general comments, which describe major changes to the user and should appear in the manual, in *version* changes (e.g., 1.1#2 to 1.2#0). Selecting the appropriate options in `xsbdoc` then allows including in the manual the version changes but not the patch changes (which might on the other hand be included in an *internals manual*).

1.5.3 Cleaning up the documentation directory

Typing the command:

```
?- make_distclean.
```

deletes all intermediate files and the generated `.texic` files, leaving only the targets (i.e., the `.ps`, `.dvi`, `.ascii`, `.html`, etc. files). This is the option normally used when building software distributions in which the manuals come ready made in the distribution itself and will not need to be generated during installation.

1.5.4 Ensuring Compatibility with All Supported Target Formats

`xsbdoc` allows manuals to be generated in several different formats. Because these formats each have widely different requirements it is sometimes a little tricky to get things to work successfully for all formats. The following recommendations may help:

- The GNU info format requires all *nodes* (chapters, sections, etc.) to have different names. This is ensured by `xsbdoc` for the automatically generated sections (by appending the module or file name to all section headings). However, care must be taken when writing section names manually to make them different. For example, use “`xsbdoc usage`” instead of simply “`Usage`”, which is much more likely to be used as a section name in another file being documented.
- Also due to a limitation of the `info` format, (used in generating formatted ASCII files) do not use `:` or `,` or `--` in section, chapter, etc. headings.
- The character “`_`” in names may sometimes give problems in indices, since current versions of `texinfo` do not always handle it correctly.

1.5.5 Generating auxiliary files (e.g., READMEs)

Using `xsbdoc` it is often possible to use a common source for documentation which should appear in several places. For example, if a file `INSTALL.xsbdoc` contains text (with `xsbdoc` formatting commands) describing an application. This text can be included in a section of the main file documentation as described above:

```
:- comment(module, ""
...
@section{Installation instructions}
@include{INSTALL.xsbdoc}
...
"").
```

At the same time, this text can be used to generate a nicely formatted `INSTALL` file in `ascii`. To this end, an `INSTALL.P` file as follows can be constructed:

```
:- comment(title,"Installation instructions").
:- comment(module,"@include{INSTALL.xsbdoc}").
```

Then, the ascii INSTALL file can be generated by simply running `xsbdoc(format,ascii)` using a file `format.P` with the fact `xsbdoc_main('INSTALL')`.

1.6 Troubleshooting

These are some common errors which may be found using `xsbdoc` and the usual fix:

- Messages of the type:

```
! No room for a new @write .
```

while converting from `.texi` to `.dvi` (i.e., while running `tex`). These messages are `tex`'s way of saying that an internal area (typically for an index) is full. This is normally because more indices were selected in the `INDICES` variable of the `SETTINGS` file than the maximum number supported by the installed version of `tex/ texinfo` installations, as mentioned in Section 1.3 [Generating a manual], page 3. The easiest fix is to reduce the number of indices generated. Alternatively, it may be possible to recompile your local `tex/ texinfo` installation with a higher number of indices.

- Missing links in `info` files (a section which exists in the printed document cannot be accessed in the on-line document) can be due to the presence of a colon (:), a comma (,), a double dash (--), or other such separators in a section name. Due to limitations of `info` section names cannot contain these symbols.
- Menu listings in `info` which *do not work* (i.e., the menu listings are there, but they cannot be followed): see if they are indented. In that case it is due to an `itemize` or `enumerate` which was not closed.

1.7 Known bugs and planned improvements (xsbdoc1)

- Variable names are not propagated in head patterns of `:- pred` assertions..
- Variable names are not propagated in head patterns of `:- pred` assertions..
- `load_dyn/1`, `load_dynnc/1` directives should be doc'd at interface of module
- `load_dyn/1`, `load_dynnc/1` directives should be doc'd at interface of module
- Special indexing directives (e.g. `tries`) are not documented in predicate level documentation. Other directives may also be added to documentation
- Special indexing directives (e.g. `tries`) are not documented in predicate level documentation. Other directives may also be added to documentation
- Special indexing directives (e.g. `tries`) are not documented in predicate level documentation. Other directives may also be added to documentation
- Have not tested out `_doc` files; that more than one bibfile can be used; that image files are appropriately read.
- Have not tested out `_doc` files; that more than one bibfile can be used; that image files are appropriately read.
- Have not tested out `_doc` files; that more than one bibfile can be used; that image files are appropriately read.

1.8 Version/Change Log (xsbdoc1)

Version 0.1#1 (2001 / 11 / 1)

Celebrating the first version with a brand-new comment!

2 Enhancing Documentation with Machine-Readable Comments

Author(s): Manuel Hermenegildo.

This defines the admissible uses of the `comment/2` declaration (which is used mainly for adding machine readable comments to programs), the formatting commands which can be used in the text strings inside these comments, and some related properties and data types. These declarations are ignored by the compiler in the same way as classical comments. Thus, they can be used to document the program source in place of (or in combination with) the normal comments typically inserted in the code by programmers. However, because they are more structured and they are machine-readable, they can also be used to generate printed or on-line documentation automatically, using the `lpdoc` automatic documentation generator. These *textual comments* are meant to be complementary to the formal statements present in *assertions* (see the `assertions` library).

2.1 Usage and interface (comments)

- **Library usage:**

It is not necessary to use this library in user programs. Recognition of comments is handled automatically by `xsbdoc` and the XSB compiler.

- **Exports:**

- *Predicates:*

`comment/2`, `docstring/1`, `stringcommand/1`, `version_descriptor/1`.

2.2 Documentation on exports (comments)

comment/2:

PREDICATE

This declaration provides one of the main means for adding *machine readable comments* to programs (the other one is adding *documentation strings* to assertions).

Usage 1: `comment(title,TitleText)`

- *Description:* Provides a *title* for the library or application. When generating documentation automatically, the text in `TitleText` will be used appropriately (e.g., in the cover page as document title or as chapter title if part of a larger document). This will also be used as a brief description of the manual in tables of content, etc. There should be at most one of these declarations per module.

- *Example:*

```
:- comment(title,"Documentation-Oriented Assertions").
```

- *The following properties hold upon exit:*

`docstring(TitleText)` (undefined property)

Usage 2: `comment(subtitle,SubTitleText)`

- *Description:* Provides *subtitle* lines. This can be, e.g., an explanation of the application to add to the title, the address of the author(s) of the application, etc. Several of these declarations can appear per module, which is useful for, say, multiple line addresses.

- *Example:*

```
:- comment(subtitle,"A Reference Manual").
```

- *The following properties hold upon exit:*

```
docstring(SubTitleText) (undefined property)
```

Usage 3: `comment(author,AuthorText)`

- *Description:* Specifies the *author*(s) of the module or application. If present, the text in `AuthorText` will be placed in the corresponding chapter or front page. There can be more than one of these declarations per module. In order for author indexing to work properly, please use one author declaration per author. If more explanation is needed (who did what when, etc.) use an acknowledgements or subtitle comment.

- *Example:*

```
:- comment(author,"Alan Robinson").
```

- *The following properties hold upon exit:*

```
docstring(AuthorText) (undefined property)
```

Usage 4: `comment(ack,AckText)`

- *Description:* Provides *acknowledgements* for the module. If present, the text in `AckText` will be placed in the corresponding chapter or section. There can be only one of these declarations per module.

- *Example:*

```
:- comment(ack,"Module was written by Alan, but others helped.").
```

- *The following properties hold upon exit:*

```
docstring(AckText) (undefined property)
```

Usage 5: `comment(copyright,CopyrightText)`

- *Description:* Provides a *copyright* text. This normally appears somewhere towards the beginning of a printed manual. There should be at most one of these declarations per module.

- *Example:*

```
:- comment(copyright,"Copyright © 2001 FSF.").
```

- *The following properties hold upon exit:*

```
docstring(CopyrightText) (undefined property)
```

Usage 6: `comment(summary,SummaryText)`

- *Description:* Provides a brief global explanation of the application or library. The text in `SummaryText` will be used as the *abstract* for the whole manual. There should be at most one of these declarations per module.

- *Example:*

```
:- comment(summary,"This is a @bf{very} important library.").
```

- *The following properties hold upon exit:*
`docstring(SummaryText)` (undefined property)

Usage 7: `comment(module, CommentText)`

- *Description:* Provides the main comment text for the module or application. The text in `CommentText` will be used as the *introduction* or *main body* of the corresponding chapter or manual. There should be at most one of these declarations per module. `CommentText` may use subsections if substructure is needed.

- *Example:*

```
:- comment(module, "This module is the @lib{comments} library.").
```

- *The following properties hold upon exit:*
`docstring(CommentText)` (undefined property)

Usage 8: `comment(appendix, CommentText)`

- *Description:* Provides additional comments text for a module or application. The text in `CommentText` will be used in one of the last sections or appendices of the corresponding chapter or manual. There should be at most one of these declarations per module. `CommentText` may use subsections if substructure is needed.

- *Example:*

```
:- comment(appendix, "Other module functionality...").
```

- *The following properties hold upon exit:*
`docstring(CommentText)` (undefined property)

Usage 9: `comment(usage, CommentText)`

- *Description:* Provides a description of how the library should be loaded. Normally, this information is gathered automatically when generating documentation automatically. This declaration is meant for use when the module needs to be treated in some special way. There should be at most one of these declarations per module.

- *Example:*

```
:- comment(usage, "Do not use: still in development!").
```

- *The following properties hold upon exit:*
`docstring(CommentText)` (undefined property)

Usage 10: `comment(PredName, CommentText)`

- *Description:* Provides an introductory comment for a given predicate, function, property, type, etc., denoted by `PredName`. The text in `Text` will be used as the introduction of the corresponding predicate/function/... description. There should be at most one of these declarations per predicate, function, property, or type.

- *Example:*

```
:- comment(comment/2, "This declaration provides one of the main means for adding @concept{machine readable comments} to programs.").
```

- *The following properties hold upon exit:*
`docstring(CommentText)` (undefined property)

Usage 11: `comment(bug,CommentText)`

- *Description:* Documents a known *bug* or *planned improvement* in the module or application. Several of these declarations can appear per module. When generating documentation automatically, the text in the `Text` fields will be used as items in an itemized list of module or application bugs.

- *Example:*

```
:- comment(bug,"Comment text still has to be written by user.").
```

- *The following properties hold upon exit:*
`docstring(CommentText)` (undefined property)

Usage 12: `comment(Version,CommentText)`

- *Description:* Provides a means for keeping a *log of changes*. `Version` contains the *version number* and date corresponding to the change and `CommentText` an explanation of the change. Several of these declarations can appear per module. When generating documentation automatically, the texts in the different `CommentText` fields typically appear as items in an itemized list of changes. The emacs CIAO mode helps tracking version numbers by prompting for version comments when files are saved. This mode requires version comments to appear in reverse chronological order (i.e., the topmost comment should be the most recent one).

- *Example:*

```
:- comment(version(1*1+21,1998/04/18,15:05*01+'EST'), "Added some  
missing comments. (Manuel Hermenegildo)").
```

- *The following properties hold upon exit:*
`version_descriptor(Version)` (undefined property)
`docstring(CommentText)` (undefined property)

Usage 13: `comment(doinclude,PredName)`

- *Description:* This usage provides control over which predicates are included in the documentation. Normally, only exported predicates are documented. A declaration `:- comment(doinclude,PredName).` forces documentation for predicate `PredName` to be included even if `PredName` is not exported (and is not included in a `document_export/1` statement).

- *Example:*

```
:- comment(doinclude,p/3).
```

Usage 14: `comment(doinclude,PredName)`

- *Description:* As above, but allows the second argument of `:- comment(doinclude,...)` to be a list of predicate names.

Usage 15: `comment(hide,PredName)`

- *Description:* This usage has an effect opposite to the previous usage: it signals that an exported predicate should *not* be included in the documentation.

– *Example:*

```
:- comment(hide,p/3).
```

Usage 16: `comment(hide,PredName)`

– *Description:* A different usage which allows the second argument of `:-comment(hide,...)` to be a list of predicate names.

docstring/1:

PREDICATE

Defines the format of the character strings which can be used in machine readable comments (`comment/2` declarations) and assertions. These character strings can include certain *formatting commands*.

- All printable characters are admissible in documentation strings except “@”, “{,” and “}”. To produce these characters the following *escape sequences* should be used, respectively: @@, @{, and @}.
- In order to allow better formatting of on-line and printed manuals, in addition to normal text, certain formatting commands can be used within these strings. The syntax of all these commands is:

`@command`

(followed by either a space or { }), or

`@command{body}`

where *command* is the command name and *body* is the (possibly empty) command body.

The set of commands currently admitted can be found in the documentation for the predicate `stringcommand/1`.

stringcommand/1:

PREDICATE

Defines the set of structures which can result from parsing a formatting command admissible in comment strings inside assertions.

In order to make it possible to produce documentation in a wide variety of formats, the command set is kept small. The names of the commands are intended to be reminiscent of the commands used in the LaTeX text formatting system, except that “@” is used instead of “\.” Note that \ would need to be escaped in ISO-Prolog strings, which would make the source less readable (and, in any case, many ideas in LaTeX were taken from scribe, where the escape character was indeed @!).

The following are the currently admissible commands.

- **Indexing commands:**

The following commands are used to mark certain words or sentences in the text as concepts, names of predicates, libraries, files, etc. The use of these commands is highly recommended, since it results in very useful indices with little effort.

`@index{text}`

text will be printed in an emphasized font and will be included in the concept definition index (and also in the usage index). This

command should be used for the first or *definitional* appearance(s) of a concept. The idea is that the concept definition index can be used to find the definition(s) of a concept.

@cindex{*text*}

text will be included in the concept index (and also in the usage index), but it is not printed. This is used in the same way as above, but allows sending to the index a different string than the one that is printed in the text.

@concept{*text*}

text will be printed (in a normal font). This command is used to mark that some text is a defined concept. In on-line manuals, a direct access to the corresponding concept definition may also be generated. A pointer to the place in which the @concept command occurs will appear only in the usage index.

@pred{*predname*}

predname (which should be in functor/arity form) is the name of a predicate and will be printed in fixed-width, typewriter-like font. This command should be used when referring to a predicate (or a property or type) in a documentation string. A reference will be included in the usage index. In on-line manuals, a direct access to the corresponding predicate definition may also be generated.

@op{*operatorname*}

operatorname (which should be in functor/arity form) is the name of an operator and will be printed in fixed-width, typewriter-like font. This command should be used when referring to an operator in a documentation string. A reference will be included in the usage index. In on-line manuals, a direct access to the corresponding operator definition may also be generated.

@lib{*libname*}

libname is the name of a library and will be printed in fixed-width, typewriter-like font. This command should be used when referring to a module or library in a documentation string. A reference will be included in the usage index. In on-line manuals, a direct access to the corresponding module definition may also be generated.

@apl{*aplname*}

aplname is the name of an application and will be printed in fixed-width, typewriter-like font. This command should be used when referring to an application in a documentation string. A reference will be included in the usage index.

@file{*filename*}

filename is the name of a file and will be printed in fixed-width, typewriter-like font. This command should be used when referring to a file in a documentation string. A reference will be included in the usage index.

`@var{varname}`

varname is the name of a variable and will be formatted in an emphasized font. Note that when referring to variable names in a `pred/1` declaration, such names should be enclosed in `@var` commands for the automatic documentation system to work correctly.

- **Referencing commands:**

The following commands are used to introduce *bibliographic citations* and *references to sections, urls, email addresses*, etc.

`@cite{keyword}`

keyword is the identifier of a *bibliographic entry*. Such entry is assumed to reside in one of a number of `bibtex` files (*.bib files*). A reference in brackets ([]) is inserted in the text and the full reference is included at the end, with all other references, in an appendix. For example, `@cite{iso-prolog}` will introduce a citation to a bibliographic entry whose keyword is `iso-prolog`. The list of bibliography files which will be searched for a match is determined by the `BIBFILES` variable of the `lpdoc SETTINGS` file.

`@ref{section title}`

introduces at point a reference to the section or node *section title*, where *section title* must be the exact *text* of the section title.

`@uref{URL}`

introduces at point a reference to the *Universal Resource Locator* (i.e., a *WWW address 'URL'*).

`@uref{text}{URL}`

introduces at point a reference to the *Universal Resource Locator* URL, associated to the text *text*.

`@email{address}`

introduces at point a reference to *email address address*.

`@email{text}{address}`

introduces at point a reference to the email address *address*, associated to the text *text*.

- **Formatting commands:**

The following commands are used to format certain words or sentences in a special font, build itemized lists, introduce sections, include examples, etc.

`@comment{text}`

text will be treated as a *comment* and will be ignored.

`@begin{itemize}`

marks the beginning of an *itemized list*. Each item should be in a separate paragraph and preceded by an `@item` command.

`@item`

marks the beginning of a new *item in an itemized list*.

- `@end{itemize}`
marks the end of an itemized list.
- `@begin{enumerate}`
marks the beginning of an *enumerated list*. Each item should be in a separate paragraph and preceded by an `@item` command.
- `@end{enumerate}`
marks the end of an enumerated list.
- `@begin{description}`
marks the beginning of a *description list*, i.e., a list of items and their description (this list describing the different allowable commands is in fact a description list). Each item should be in a separate paragraph and contained in an `@item{itemtext}` command.
- `@item{itemtext}`
marks the beginning of a *new item in description list*, and contains the header for the item.
- `@end{description}`
marks the end of a description list.
- `@begin{verbatim}`
marks the beginning of *fixed format text*, such as a program example. A fixed-width, typewriter-like font is used.
- `@end{verbatim}`
marks the end of formatted text.
- `@begin{cartouche}`
marks the beginning of a section of text in a *framed box*, with round corners.
- `@end{cartouche}`
marks the end of a section of text in a framed box.
- `@section{text}`
starts a *section* whose title is *text*. Due to a limitation of the `info` format, do not use `:` or `-` or `,` in section, subsection, title (chapter), etc. headings.
- `@subsection{text}`
starts a *subsection* whose title is *text*.
- `@footnote{text}`
places *text* in a *footnote*.
- `@today`
prints the current *date*.
- `@hfill`
introduces horizontal filling space (may be ignored in certain formats).

`@bf{text}`

text will be formatted in *bold face* or any other *strong face*.

`@em{text}`

text will be formatted in *italics face* or any other *emphasis face*.

`@tt{text}`

text will be formatted in a *fixed-width font* (i.e., *typewriter-like font*).

`@key{key}`

key is the identifier of a *keyboard key* (i.e., a letter such as **a**, or a special key identifier such as RET or DEL) and will be formatted as ␣ or in a fixed-width, typewriter-like font.

`@sp{N}`

generates *N blank lines* of space. Forces also a paragraph break.

`@p`

forces a *paragraph break*, in the same way as leaving one or more blank lines.

`@noindent`

used at the beginning of a paragraph, states that the first line of the paragraph should not be indented. Useful, for example, for *avoiding indentation* on paragraphs that are continuations of other paragraphs, such as after a verbatim.

- **Accents and special characters:**

The following commands can be used to insert *accents* and *special characters*.

`@' {o}` ⇒ ò

`@' {o}` ⇒ o

`@~ {o}` ⇒ ô

`@. . {o}` ⇒ ö

`@~ {o}` ⇒ õ

`@= {o}` ⇒ õ

`@. {o}` ⇒ ó

`@u {o}` ⇒ ŏ

`@v {o}` ⇒ ǒ

`@H {o}` ⇒ Ǔ

`@t {oo}` ⇒ õö

`@c {o}` ⇒ ç

`@d {o}` ⇒ ç

`@b {o}` ⇒ ç

@oe	⇒	œ
@OE	⇒	Œ
@ae	⇒	æ
@AE	⇒	Æ
@aa	⇒	å
@AA	⇒	Å
@o	⇒	ø
@O	⇒	Ø
@l	⇒	ł
@L	⇒	Ł
@ss	⇒	ß
@?	⇒	¿
@!	⇒	¡
@i	⇒	ı
@j	⇒	ĵ
@copyright	⇒	©
@iso	⇒	◻•ISO•◻
@bullet	⇒	•
@result	⇒	⇒

- **Inclusion commands:**

The following commands are used to include code or strings of text as part of documentation. The latter may reside in external files or in the file being documented. The former must be part of the module being documented. There are also commands for inserting and scaling images.

@include{*filename*}

the contents of *filename* will be included in-line, as if they were part of the string. This is useful for common pieces of documentation or storing in a separate file long explanations if they are perceived to clutter the source file.

@includeverbatim{*filename*}

as above, but the contents of the file are included verbatim, i.e., commands within the file are not interpreted. This is useful for including code examples which may contain @'s, etc.

@includefact{*factname*}

it is assumed that the file being documented contains a fact of the predicate *factname/1*, whose argument is a character string. The

contents of that character string will be included in-line, as if they were part of the documentation string. This is useful for *sharing pieces of text* between the documentation and the running code. An example is the text which explains the *usage of a command* (options, etc.).

`@includedef{predname}`

it is assumed that the file being documented contains a definition for the predicate *predname*. The clauses defining this predicate will be included in-line, in verbatim mode, as if they were part of the documentation string.

`@image{epsfile}`

including an image at point, contained in file *epsfile*. The *image file* should be in *encapsulated postscript* format.

`@image{epsfile}{width}{height}`

same as above, but *width* and *height* should be integers which provide a size (in points) to which the image will be scaled.

version_descriptor/1:

PREDICATE

A structure denoting a complete version description:

```
version_descriptor(version('Version','Date','Time')) :-
    version_number('Version'),
    ymd_date('Date'),
    time_struct('Time').
```

```
version_descriptor(version('Version','Date')) :-
    version_number('Version'),
    ymd_date('Date').
```

2.3 Documentation on internals (comments)

version_number/1:

PREDICATE

Version is a structure denoting a complete version number (major version, minor version, and patch number):

```
version_number('Major' * 'Minor' + 'Patch') :-
    int('Major'),
    int('Minor'),
    int('Patch').
```

ymd_date/1:

PREDICATE

A Year/Month/Day structure denoting a date:

```
ymd_date('Y' / 'M' / 'D') :-  
    int('Y'),  
    int('M'),  
    int('D').
```

time_struct/1:

PREDICATE

A structure containing time information:

```
time_struct('Hours' : 'Minutes' * 'Seconds' + 'TimeZone') :-  
    int('Hours'),  
    int('Minutes'),  
    int('Seconds'),  
    atm('TimeZone').
```

3 The XSB Assertion Library

This module is under development and is subject to change. Assertions in this style will be documented by `xsbdoc`, but no type checking is done yet by the compiler.

The `assertions` module has many of the same goals of the CIAO assertions package, but is much simpler in syntax. Currently all assertions (apart from machine readable comments) are specified via the compiler directive

```
:- pred HeadPattern :: AssertionBody.
```

where `head_pattern(HeadPattern)` and `AssertionBody` are both true. The idea is that each such declaration defines a usage of a predicate – a call pattern and an associated success pattern. The exact syntax is described in documentation for the predicates `head_pattern/1` and `assertion_body/1`.

Such patterns are certainly useful for documentation, and `xsbdoc` includes these patterns in the documentation it generates (see Chapter 1 [Introduction], page 2 for information on how to control formatting of such documentation). However, such information can also be useful for compilation tools, such as a run-time test generator (under development) or static analysis.

3.1 Usage and interface (`assertions_props`)

- **Exports:**

- *Predicates:*

```
assertion_body/1, head_pattern/1, assrt_status/1.
```

3.2 Documentation on exports (`assertions_props`)

`assertion_body/1:`

PREDICATE

This predicate succeeds if its argument is true. In XSB, assertion bodies are much simpler than in Ciao. An assertion body is a list of up to 5 terms. These are:

- `call_prop(Comma_list)` which contains a comma list of predicates which are true of given arguments when the predicate is called for a given usage of a predicate to be true.
- `success_prop(Comma_list)` which contains a comma list of predicates which must be true of given arguments when the predicate succeeds for this usage.
- `global_prop(Comma_list)` which contains a comma list of predicates which must be true of given arguments at any point during an execution.
- `compat_prop(Comma_list)` which contains a comma list of predicates which must be true for this usage that do not fit into one of the other properties, for instance that the this usage of the predicate does not fail.

- `comment(Docstring)` which contains a string of type `docstring/1` which describes this usage.

An example is of an assertion body occurs in the following predicate:

```
:- pred foo(X,Y) ::
    [call_prop((nonvar(X),var(Y))),
     success_prop((ground(X),ground(Y)))]
```

head_pattern/1:

PREDICATE

A head pattern can be a predicate name (functor/arity) (`predname/1`) or a term. Thus, both `p/3` and `p(A,B,C)` are valid head patterns. In the case in which the head pattern is a term, each argument of such a term can be:

- A variable. This is useful in order to be able to refer to the corresponding argument positions by name within properties and in comments. Thus, `p(Input,Parameter,Output)` is a valid head pattern.
- A term. In this case, the fact that a given argument unifies with a particular term is a global property of the usage. An example is the pattern

```
comment(module,Docstring)
```

which indicates that the first argument is always unifiable with the atom `module`.

assrt_status/1:

PREDICATE

The types of assertion status. They have the same meaning as the program-point assertions, and are as follows:

```
assrt_status(trust).
```

```
assrt_status(checked).
```

```
assrt_status(check).
```

```
assrt_status(false).
```

```
assrt_status(true).
```

. TLS No checking is yet implemented in XSB.

References

- [BCC] F. Bueno, D. Cabenza, M. Carro, M. Hermenegildo, P. L'opez-Garc'ia, and G. Puebla.
The ciao prolog system, reference manual.
Technical report, School of Computer Science, Technical University of Madrid.
Available from <http://www.clip.dia.fi.upm.es/>.

Predicate Definition Index

A

assertion_body/1	23
assrt_status/1	24

C

comment/2	11
-----------------	----

D

docstring/1	15
-------------------	----

H

head_pattern/1	24
----------------------	----

S

stringcommand/1	15
-----------------------	----

T

time_struct/1	22
---------------------	----

V

version_descriptor/1	21
version_number/1	21

Y

ymd_date/1	21
------------------	----

Operator Definition Index

(Index is empty)

Concept Definition Index

•		
.bib files	17	
@		
@! command	20	@j command
@' command	19	@key command
@. command	19	@l command
@.. command	19	@L command
@= command	19	@lib command
@? command	20	@noindent command
@~ command	19	@o command
@' command	19	@O command
@~ command	19	@oe command
@aa command	20	@OE command
@AA command	20	@op command
@ae command	20	@p command
@AE command	20	@pred command
@apl command	16	@ref command
@b command	19	@result command
@begin{cartouche} command	18	@section command
@begin{description} command	18	@sp command
@begin{enumerate} command	18	@ss command
@begin{itemize} command	17	@subsection command
@begin{verbatim} command	18	@t command
@bf command	19	@today command
@bullet command	20	@tt command
@c command	19	@u command
@cindex command	16	@uref command
@cite command	17	@v command
@comment command	17	@var command
@concept command	16	
@copyright command	20	A
@d command	19	A4 paper
@em command	19	abstract
@email command	17	accents
@end{cartouche} command	18	acknowledgements
@end{description} command	18	appendix
@end{enumerate} command	18	assertions
@end{itemize} command	18	author
@end{verbatim} command	18	avoiding indentation
@file command	16	
@footnote command	18	B
@H command	19	bibliographic citations
@hfill command	18	bibliographic entry
@i command	20	bibtex
@image command	21	blank lines
@include command	20	bold face
@includedef command	21	bug
@includefact command	20	
@includeverbatim command	20	C
@index command	15	comment
@iso command	20	component files
@item command	17, 18	copyright

D

date 18
 description list 18
 documentation strings 11

E

email address 17
 email addresses 17
 emphasis face 19
 encapsulated postscript 21
 enumerated list 18
 escape sequences 15

F

fixed format text 18
 fixed-width font 19
 footnote 18
 formatting commands 15
 framed box 18

I

image file 21
 images, inserting 21
 images, scaling 21
 including a predicate definition 21
 including an image 21
 including code 20
 including files 20
 including images 20
 indentation, avoiding 19
 internals manual 8
 introduction 13
 italics face 19
 item in an itemized list 17
 itemized list 17

K

keyboard key 19

L

letter size paper 6
 literate programming 7
 log of changes 14

M

machine readable comments 11
 main body 13
 main file 3
 module comment 13

N

new item in description list 18

P

page numbering, changing 5
 page size, changing 5
 page style, changing 5
 paragraph break 19
 planned improvement 14

R

references 17

S

section 18
 sections 17
 sharing pieces of text 21
 space, extra lines 19
 spcae, horizontal fill 18
 special characters 19
 strong face 19
 subsection 18
 subtitle 12
 syntax of formatting commands 15

T

texinfo files 2
 textual comments 11
 thesis-like style 6
 title 11
 typewriter-like font 19

U

Universal Resource Locator 17
 URL 17
 urls 17
 usage of a command 21

V

verbatim text 18
 version number 14

W

WWW address 17

Global Index

This is a global index containing pointers to places where concepts, predicates, modes, properties, types, applications, etc., are referred to in the text of the document. Note that due to limitations of the `info` format unfortunately only the first reference will appear in online versions of the document.

.		@includefact command.....	20
.bib files.....	17	@includeverbatim command.....	20
Q		@index command.....	15
@! command.....	20	@iso command.....	20
@' command.....	19	@item command.....	17, 18
@. command.....	19	@j command.....	20
@.. command.....	19	@key command.....	19
@= command.....	19	@l command.....	20
@? command.....	20	@L command.....	20
@^ command.....	19	@lib command.....	16
@' command.....	19	@noindent command.....	19
@~ command.....	19	@o command.....	20
@aa command.....	20	@O command.....	20
@AA command.....	20	@oe command.....	20
@ae command.....	20	@OE command.....	20
@AE command.....	20	@op command.....	16
@apl command.....	16	@p command.....	19
@b command.....	19	@pred command.....	16
@begin{cartouche} command.....	18	@ref command.....	17
@begin{description} command.....	18	@result command.....	20
@begin{enumerate} command.....	18	@section command.....	18
@begin{itemize} command.....	17	@sp command.....	19
@begin{verbatim} command.....	18	@ss command.....	20
@bf command.....	19	@subsection command.....	18
@bullet command.....	20	@t command.....	19
@c command.....	19	@today command.....	18
@cindex command.....	16	@tt command.....	19
@cite command.....	17	@u command.....	19
@comment command.....	17	@uref command.....	17
@concept command.....	16	@v command.....	19
@copyright command.....	20	@var command.....	17
@d command.....	19		
@em command.....	19	A	
@email command.....	17	A4 paper.....	5
@end{cartouche} command.....	18	abstract.....	12
@end{description} command.....	18	accents.....	19
@end{enumerate} command.....	18	acknowledgements.....	12
@end{itemize} command.....	18	address.....	17
@end{verbatim} command.....	18	appendix.....	13
@file command.....	16	application.....	2
@footnote command.....	18	assertion_body/1.....	23
@H command.....	19	assertions.....	2, 11, 23
@hfill command.....	18	assrt_status/1.....	23, 24
@i command.....	20	author.....	12
@image command.....	21	author indexing.....	12
@include command.....	20	avoiding indentation.....	19
@includedef command.....	21		

B

bibliographic citations	17
bibliographic entry	17
bibtex	17
blank lines	19
bold face	19
bug	14

C

comment	17
comment/2	7, 11, 15
comments	2
component files	3
copyright	12

D

date	18
description list	18
doc	3
docstring/1	11, 15, 24
document_export/1	6
document_import/1	6
documentation strings	11
dvips	1, 6
dynamic/1	7

E

emacs CIAO mode	14
email address	17
email addresses	17
emphasis face	19
encapsulated postscript	21
enumerated list	18
escape sequences	15
export/1	6

F

fixed format text	18
fixed-width font	19
footnote	18
format file	2
formatting commands	2, 11, 15
framed box	18

G

GNU general public license	1
----------------------------	---

H

head pattern	24
head_pattern/1	23, 24

I

image file	21
images, inserting	21
images, scaling	21
import/1	6
including a predicate definition	21
including an image	21
including code	20
including files	20
including images	20
indentation, avoiding	19
info	8, 9, 18
INSTALL.xsbdoc	8
internals manual	8
introduction	13
italics face	19
item in an itemized list	17
itemized list	17

K

keyboard key	19
--------------	----

L

latex	1
LaTeX	15
letter size paper	6
library	2
library_directory/1	4
literate programming	7
log of changes	14
lpdoc	1, 2, 4, 11, 17

M

machine readable comments	11
main body	13
main file	3
makeinfo	6
module comment	13

N

new item in description list	18
------------------------------	----

O

op/3	7
------	---

P

page numbering, changing 5
 page size, changing 5
 page style, changing 5
 paragraph break 19
 planned improvement 14
 pred/1 1, 17
 pred/2 7
 predname/1 24
 Prolog 2
 Prolog source files 2
 ps2pdf 6

R

references 17

S

scribe 15
 section 18
 sections 17
 SETTINGS 17
 sharing pieces of text 21
 space, extra lines 19
 spcae, horizontal fill 18
 special characters 19
 stringcommand/1 11, 15
 strong face 19
 subsection 18
 subtitle 12
 syntax of formatting commands 15

T

tex 6, 9
 texi2html 6
 texinfo 2, 3, 5, 8, 9
 texinfo files 2
 textual comments 11
 thesis-like style 6
 time_struct/1 22

title 11
 typewriter-like font 19

U

Universal Resource Locator 17
 URL 17
 urls 17
 usage 23
 usage of a command 21
 use_variant_tabling/1 7

V

verbatim text 18
 version number 14
 version_descriptor/1 11, 21
 version_number/1 21

W

WWW address 17

X

xdvi 6
 XSB 1, 2
 xsbdoc 1, 2, 3, 4, 6, 7, 8, 9, 23
 xsbdoc/2 6
 xsbdoc_bibfile/1 4
 xsbdoc_component/1 3
 xsbdoc_index/1 5
 xsbdoc_main/1 3, 6
 xsbdoc_option/1 4
 xsbdoc_papertype/1 5
 xsbdoc_startpage/1 5

Y

yap 6
 ymd_date/1 21