aggedbottom

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

Table of Contents

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.

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 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.

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('-noversion')`
  : 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 9).

  `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 ⟨undefined⟩).

`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 ⟨undefined⟩).

`xsbdoc_option('-v')`
: Generates verbose output and can be useful for debugging document speficications.

- 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 ⟨undefined⟩ [The XSB Assertion Library], page ⟨undefined⟩).

  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), bbugs, 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 ⟨undefined⟩.

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..
- load_dyn/1, load_dync/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
- 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!