SoarDoc User's Manual

Dave Ray
ray@soartech.com
October 16, 2003

Introduction

SoarDoc is an embedded metadata documentation format and tool for Soar. This format facilitates the automatic generation of flexible project documentation in the spirit of Doxygen, Javadoc, and other documentation generation tools.

The SoarDoc tool takes as input a set of Soar source files and produces as output a set of HTML documentation for the files. The output provides several distinct "views" onto the documentation. For example, the documentation can be browsed by source file, problem-space, operator, or by a complete alphabetical index of production names.

Getting Started

This document assumes you already have the SoarDoc source code installed on your system. SoarDoc requires Python version 2.2, available from www.python.org. SoarDoc is not compatible with earlier versions of Python. To find out which version of Python you have type "python –V" at the command line.

The file **soardoc.py** in the **src/** directory is the main script for SoarDoc. To run SoarDoc, you must pass **soardoc.py** as an argument to the python interpreter, like this:

```
$ /path/to/python /path/to/soardoc/src/soardoc.py
```

Running the **makebat.py** script in SoarDoc's root directory will generate a shell script called **soardoc** (**soardoc.bat** on Windows systems) that simplifies this process. For the rest of this document, it is assumed that this script is available on the system path.

SoarDoc will generate documentation for any directory tree of Soar source files, even if they haven't been commented yet. Once you have the SoarDoc executable, try this:

```
$ cd /path/to/your/source/code
$ soardoc
```

This will create a new directory **html/** in the current directory containing a set of HTML files giving a basic overview of your system. If your Soar productions are already documented by preceding comments, try these commands instead:

```
$ cd /path/to/your/source/code
$ soardoc -C UseExistingComments=1
```

SoarDoc will generate the same documentation as before, but it will include your existing comments in the production documentation.

To further customize SoarDoc's output, see the Configuration Options section.

The next section describes how to document your code to make the most out of SoarDoc.

Documenting the Code

Special Documentation Blocks

Special documentation blocks are simply specially formatted Soar comments. The general format is as follows:

```
##!
#
#  ... text ...
#
#
```

The block ends at the first line that does not begin with a # character. The contents of the blocks can be any valid SoarDoc command, as well as HTML, which will be passed through to the output unchanged. HTML tags allow the user to easily format text, create bulleted lists, insert hyperlinks to external data, include images, etc.

Note: Since Soar comments are really Tcl comments, extra care has to be taken with { and } characters. They must be balanced even though they appear in comments.

Documenting a File

A documentation block whose first command is **@file** provides documentation for a file (usually the file in which is appears). Here is a sample for a file:

```
##!
# @file
# @brief A brief description of the file's purpose.
#
# A more detailed, possibly
# multi-line description of the file's purpose.
# This text can be <b>formatted</b> with normal
# HTML tags.
#
# @project MyProject
#
# @kernel 8.3 Works
# @kernel 8.4 Comments about kernel 8.4...
#
# @created ray 20030120 MyProject
```

```
# @modified ray 20030120 MyProject
#           Added new production
# @modified [1] ray 20030127 MyProject
#         Made some general changes...
```

The **@file** command specifies the name of the file that is being documented (use relative paths as necessary). If the name is not given, then the name of the current file is assumed. The **@brief** command specifies a short description of the file, used for indices and tables of content in the output. The main body of text is a more detailed description of the file and its contents. The **@project** command specifies a list of projects to which the file belongs. The **@kernel** command specifies a list of Soar kernel version numbers with which this file is compatible.

For more details on the **@created** and **@modified** commands, see the Documenting Change History section.

Documenting a Production

A documentation block that precedes a production definition provides documentation for that production. Here is an example:

```
##!
# @production test0
# @brief A brief description of the production.
#
# A more detailed, possibly
# multi-line description of the production.
# This text can be <b>formatted</b> with normal
# HTML tags.
#
# @kernel 8.3 Works
# @kernel 8.4 Comments about kernel 8.4...
#
# @problem-space test0
# @type elaboration
# @created ray 20030120
# @modified [1] More specific description of
#         change [1]
#
sp {test0
   (state <s> ^problem-space.name test0
           ^superstate.operator.point <r>)
-->
   (<s> ^point <r>)
   (<r> ^sub-achieved *yes*)
}
```

Here most of the commands are identical to those found in a file-scope documentation block. Since this documentation block immediately precedes a production definition, then the **@production** command is not entirely necessary, but if the production was located elsewhere, or if it was generated by, say, a call to a TCL function, the **@production** command would be required. The **@problem-space** command specifies a list of problem spaces to which the production belongs. Similarly, the **@operator** command is used to specify operators. The **@type** command indicates the production's type. See the Command Reference section for details.

In the case of the **@type**, **@kernel**, **@problem-space**, and **@operator** commands, these may be specified at file scope and then inherited by the documentation of all productions in the file. When appropriate, this will reduce the amount of repetitive typing required to adequately document a production.

For more details on the **@created** and **@modified** commands, see the Documenting Change History section.

Documenting a Problem-Space

Although not explicitly defined in the Soar code, a problem-space may be documented using a SoarDoc documentation block. Here is an example:

```
##!
# @problem-space test0
# @brief This is problem space test0
#
# Long description of problem space test0...
#
# @operator operator0
```

This block defines documentation for problem-space **test0**. The **@operator** command specifies the name of the operator that spawns this problem-state. If omitted, then the parent is assumed to be the operator with the same name as the problem-space (as is the convention in Soar programming).

Documenting an Operator

An operator can be documented in the same manner as a problem-space. Here is an example:

```
##!
# @operator operator1
# @brief This is operator1
#
# Long description of operator1...
#
# @problem-space test0
#
```

This block defines documentation for operator **operator1**. The **@problem-space** command indicates its position in the operator hierarchy by specifying the names of the problem-spaces in which the operator may apply.

Documenting Change History

Using the **@created** and **@modified** commands, the change history of a documentation block (and the object it documents) can be recorded. Each command encodes who made the change, the date, the project the change affects, and any developer comments.

By using a special reference parameter, related changes in multiple parts of a file can be grouped together. For further details and example usage see the **@created** and **@modified** command documentation in the General Commands section.

Main Page Documentation

The main documentation page can be customized by inserting a documentation block as follows:

```
##!
# @mainpage Title of Custom Main page
#
# <h1> Introduction </h1>
# Here is some kind of introduction…
# <h2>Section 1</h1>
# Some more stuff here.
#
```

The body of a **@mainpage** block can be arbitrary HTML. It may also include **@ref** commands to reference other areas of the documentation.

Structural Commands

Groups

Structural documentation commands allow for more granular control over the final output of SoarDoc. Files and productions can be arbitrarily grouped, creating a hierarchical organization of documentation. For example, productions and files could be hierarchically grouped by operator.

Here is a grouping example:

```
##!
# @group group1
#
# @brief A brief description of group1
#
# Detailed description of group1.
```

```
... Meanwhile, in another file ...

##!
# @file
# @brief A file.
#
# Description of file...
#
# @ingroup group1
#
# ...
```

This example defines a group called **group1**. The **@ingroup** command is used to assign a documentation block to a particular group. A documentation block may be assigned to multiple groups.

References

Using the **@ref** command, inter-block links can be created:

```
##!
#
# This is a description that has a
# link @ref prod:bowtie*elaborate*state*problem-space
# to a production.
```

This effect could also be achieved by embedding HTML anchors into the documentation blocks, but this would be an assumption about the file layout of the final HTML output.

Ignoring Soar Code

Using the **@start-no-soardoc** and **@end-no-soardoc** commands, a section of a Soar source file can be ignored by SoarDoc. Here is an example usage:

```
##! @start-no-soardoc

source load.soar
... other code and productions that should be ignored

##! @end-no-soardoc
```

Datamap Generator Integration

SoarDoc can use the XML files generated by the Datamap Generator (dmgen) to produce browsable, graphical representations of problem-spaces and operators. Graph generation requires an installation of the latest AT&T GraphViz tools.

GraphViz is available from http://www.research.att.com/sw/tools/graphviz/

The procedure is as follows:

- Run **dmgen** on your productions to generate a datamap, using the XML output mode. *See dmgen.doc for details of how to do this.*
- In your SoarDoc config file (soardocfile):
  - Set SoarDoc's XmlDatamap parameter to the directory where the XML files are stored.
  - Set SoarDoc's ShowDatamaps parameter to **1**.
  - Set SoarDoc's DotPath and DotExeName parameters to point to the DOT executable in your GraphViz installation.
- Run SoarDoc

Note that graphs (and HTML pages) will only be generated for problem-spaces and operators that have been documented. Running SoarDoc in autodoc mode with the XmlDatamap parameter set will generate stub documentation for all problem-spaces and operators that dmgen identifies. See the Overriding Configuration Parameters

The **–C** command-line option can override configuration parameters specified in the configuration file. The basic syntax of the command is:

```
$ soardoc –C NameOfParameter=ValueOfParameter
```

If **ValueOfParameter** is a string value, it must be enclosed in single quotes, like this:

```
$ soardoc –C OutputDirectory='output'
```

On Linux, it is necessary to enclose these options in double quotes to prevent the shell from stripping out the single quotes. On Linux, the above example becomes:

```
$ soardoc –C "OutputDirectory='output'"
```

Unfortunately, for all command-line examples in this manual, these double quotes must be added to use them in Linux.
Existing Soar Code section below.

The image format used by DOT images is controlled by the **DotImageFormat** parameter. Since some datamaps can be very large and **gif** and **jpeg** formats have image size limits, it is recommended, when possible, that you use the **png** format.

Configuration Options

Command-Line Options

Details of SoarDoc command-line options can be found by running SoarDoc with the **–h** option. For example:

```
$ soardoc -h
```

Configuration File

SoarDoc is currently implemented in Python and for convenience uses Python to specify its configuration file. When SoarDoc is run it loads default values for configuration options. Next, it checks whether an alternate configuration file has been provided with the **–f** command-line option. If so, the file is loaded. Otherwise SoarDoc looks for a file called **soardocfile** in the current directory and, if it exists, it is loaded. This behavior is similar to that of Doxygen (doxyfile) or make (makefile).

The **–g** command-line option will tell SoarDoc to print a default configuration file to standard output. Thus the following command could be used to generate **soardocfile**:

```
$ soardoc -g > soardocfile
```

When generated this file contains a complete list of configuration parameters with descriptions and default values. The comments in this file are the documentation for all of SoarDoc's configuration parameters.

Overriding Configuration Parameters

The **–C** command-line option can override configuration parameters specified in the configuration file. The basic syntax of the command is:

```
$ soardoc –C NameOfParameter=ValueOfParameter
```

If **ValueOfParameter** is a string value, it must be enclosed in single quotes, like this:

```
$ soardoc –C OutputDirectory='output'
```

On Linux, it is necessary to enclose these options in double quotes to prevent the shell from stripping out the single quotes. On Linux, the above example becomes:

```
$ soardoc –C "OutputDirectory='output'"
```

Unfortunately, for all command-line examples in this manual, these double quotes must be added to use them in Linux.

Existing Soar Code

If you have a large existing Soar system, manually entering comment blocks for every object in your system would be very tedious. SoarDoc has an **autodoc** mode which will generate stub documentation for all undocumented (no existing SoarDoc comment block) objects in the system.

autodoc mode is specified by setting the **OutputFormat** configuration parameter to **'autodoc'**. Running SoarDoc in this mode will create a file called 'soardoc.soar' in the

directory specified by the **OutputDirectory** parameter. It will contain a set of stub comment blocks. If **UseExistingComments** is set to 1, SoarDoc will include any existing comments in the stub blocks.

Since **autodoc** mode should be run relatively infrequently, it's easier to run it as a "one-liner" without having to modify your configuration file:

```
$ soardoc -C OutputFormat='autodoc' -C OutputDirectory='.'
          -C UseExistingComments=1
```

with all arguments on a single line, of course.

If you have made manual changes to soardoc.soar and then rerun SoarDoc in **autodoc** mode, your changes will not be overwritten.

autodoc mode is especially useful for generating documentation for problem-spaces and operators discovered by **dmgen**. After running **dmgen** to update your datamap, rerun SoarDoc in **autodoc** mode to update the soardoc.soar file with new problem-space and operator documentation.

Processing Soar Log (Trace) Output

SoarDoc can parse and generate an annotated HTML version of a Soar log file (the contents of the agent window, or a file generated by the 'log' command in Soar). When available, operator names are replaced with their @brief description, making the trace more understandable for a non-developer.

Here's the simplest way to do this, assuming you have a log file (log.txt) in the root directory of your source tree:

$ soardoc -L log.txt

This will generate a file html/log.html.

To change the location of the output, change the **OutputDirectory** config parameter. To generate links to SoarDoc documentation, enable the **LogUseSoarDocs** parameter. If you change the **OutputDirectory**, you will probably have to modify the **LogPathToSoarDocs** parameter as well.

See the default configuration file for many more configuration options.

Command Reference

All SoarDoc commands begin with the @ symbol followed immediately by the command's name. They may take zero or more arguments. The arguments may span more than one line, but the argument list is terminated with the next "blank" line (a single '#' and whitespace) or the next line that is started with a SoarDoc.

General Commands

The following commands may appear in the documentation block for any object: file, production, problem-space, operator, or group.

@brief <description>

A single-line description of the object used in indices and tables of content.

@desc <description>

A long, usually multi-line, description of the object. The **@desc** command is the default command for any block. Thus, the actual command name may be omitted. For example, the following two blocks are equivalent:

```
##!
# @production test0
# @brief A production.
#
# @desc Here is a long description with
# <i><b>embedded</b> HTML tags</i>
#
```

and

```
##!
# @production test0
# @brief A production.
#
# Here is a long description with
# <i><b>embedded</b> HTML tags</i>
#
```

Note that in the second block, **@desc** was omitted.

@ingroup <group name> [<group name> …]

Specifies that this object is a member of the given groups. The groups must have been previously defined with the **@group** command. This command may appear more than once in a documentation block.

@created <user> <date> <project> [Comment]

Indicates when the file was created and the user that created it. **<date>** should be in YYYYMMDD format.

@modified [[refnum]] <user> <date> <project> [Comment]

Indicates a modification to the object and the user that made the change. **<date>** should be in YYYYMMDD format. This command may appear more than once in a documentation block.

The optional **[Comment]** parameter may span multiple lines and contain arbitrary HTML as well as **@ref** commands.

[refnum] is an optional first parameter which must take the form of an integer enclosed in square brackets. When present at file scope, this creates a general modification note that other **@modified** commands in the file can reference. When it appears at any other scope, then **<user>**, **<date>**, and **<project>** are inherited from the file **@modified** command with the same **[refnum]** and are not required. Only a comment is necessary. Using this mechanism, several changes throughout a file may be grouped together.

Here is an example of this usage:

At file scope…

```
##!
# @file
#
# @modified [1] ray 20030127 MyProject
#    Made several related changes...
```

At production scope…

```
##!
# @production
#
# @modified [1] More details for change [1]
```

At another production scope…

```
##!
# @production
#
# @modified [1] Even more details for change [1]
```

Note that in the production scopes, the user, date and project are omitted when referring to a more general modification. Also, using change references in this manner allows the generation of reports where a change and all affected objects can be grouped together in a single location.

The name and date fields could probably be automatically filled in by CVS keyword substitutions, but this may cause unwanted conflicts, so care should be taken.

@devnote [Comments]

Developer-centric notes. These can be enabled/disabled in the generated documentation depending upon the target audience. **[Comments]** may be arbitrary HTML possibly spanning multiple lines and containing **@ref** commands.

@todo [Comments]

All instances of the **@todo** command are collected together to form a todo list in the generated documentation. **[Comments]** may be arbitrary HTML possibly spanning multiple lines and containing **@ref** commands.

File-scope Commands

The following commands may appear in the documentation block for a file.

@file [<name>]

Required. Indicates that this documentation block documents a file. **<name>** indicates the name of the file being documented. If **<name>** is omitted, then the name of the current file is used. This command must be the first command in the block.

@project <name> [<name> …]

A list of projects to which this file belongs.

@kernel <version> [Comment]

Indicates compatibility (syntactic and semantic) with a particular version of the Soar kernel. This command may appear more than once in a documentation block to indicate compatibility with multiple kernels.

@problem-space <problem-space name> [<problem-space name> …]

Indicates the default problem-spaces used by productions in this file if they don't specify one. This command may appear more than once in a documentation block.

@operator <operator name> [<operator name> …]

Indicates the default operators used by productions in this file if they don't specify one. This command may appear more than once in a documentation block.

@type <type>

Default value for **@type** command at production scope.

Production-scope Commands

The following commands may appear in the documentation block for a production.

@production <name>

Indicates that this documentation block documents the named production. This command is optional if the block is immediately followed by the production it documents. If used this command must be the first in the block.

@kernel <version> [Comment]

Indicates compatibility (syntactic and semantic) with a particular version of the Soar kernel. This command may appear more than once in a documentation block to indicate compatibility with multiple kernels.

If this command is omitted, then the production inherits the **@kernel** commands used in the file-scope documentation block.

@problem-space <problem-space name> [<problem-space name> …]

Indicates a list of problem-spaces with which this production is associated. This command may appear more than once in a documentation block.

If this command is omitted, then the production inherits the **@problem-space** commands used in the file-scope documentation block.

@operator <operator name> [<operator name> …]

Indicates a list of operators with which this production is associated. This command may appear more than once in a documentation block.

If this command is omitted, then the production inherits the **@operator** commands used in the file-scope documentation block.

@type <type>

Indicates the purpose of the production. The suggested values for the **<type>** parameter are:

<type>	**Description**
elaboration	
state-elaboration	
operator-elaboration	
proposal	
application	
termination	In Soar 7 there are productions which terminate an operator by asserting a reconsider preference (@).
selection	These productions specify operator preferences.
persistent	These productions are o-supported elaborations (tests that an operator exists, but doesn't test the name of the operator).
suggest-proposal	

If this command is omitted then its value is inherited from the **@type** command at file scope if one exists.

Hint: SoarDoc does not enforce this list of production types so you are free to invent your own and SoarDoc will happily display them as well as use them in the "Productions by type" index.

Problem-Space-scope Commands

@problem-space <name>

Required. Name of the problem-space. This command must be the first command in the block.

@operator <name>

Name of the parent operator in the goal hierarchy. If omitted then the parent operator is taken to be the operator with the same name as the problem-space.

Operator-scope Commands

@operator <name>

Required. Name of the operator. This command must be the first command in the block.

@problem-space <name> [<name> …]

List of problem-space names in which this operator may be selected, i.e. its parents in the goal hierarchy.

Main Page Commands

@mainpage <title>

Required. Title used for the mainpage.

Group Commands

@group <name> [group title]

Define a group. The name of the group must not exist yet. This command must be the first command in the block.

Structural Commands

@ref type:<anchor name>

Refer to another documentation object

The **type** modifier allows a particular named object to be specified. Here is an example that references a production:

```
#
# Please see @ref prod:name*of*production
```

```
# for more info.
#
```

This will production a hyperlink to the documentation for production **name*of*production**.

type	Description
file	A link to file documentation
prod	A link to production documentation
ps	A link to problem-space documentation
op	A link to operator documentation
group	A link to group documentation

@start-no-soardoc and @end-no-soardoc

Used in pairs around sections of code that should be ignored by SoarDoc

Trouble-Shooting

Here are problems that may arise while using SoarDoc:

Where I expect to see datamap graphs, I see gibberish like:

digraph G { node [label="\N"]; graph [bb="0,0,0,0"]; ...

You are using an old version of dot that doesn't support output of client-side image maps (dot option -Tcmap). Get the latest version from http://www.research.att.com/sw/tools/graphviz/.

I'm running autodoc mode, but I can't find the generated soardoc.soar file?

Output is always written to the directory indicated by the **OutputDirectory** parameter which defaults to '**html'**. To quickly run in autodoc mode, outputting to the current directory without modifying your config file try:

```
$soardoc -C OutputFormat='autodoc' -C OutputDirectory='.'
```

I have generated a datamap with dmgen and set all of the related config options (XmlDatamap, etc) but no problem-spaces or operators show up in my generated docs.

The reason you can't see them in SoarDoc's output is that none of them are documented in the code. The quickest way to get them documented is:

```
$ soardoc -C OutputFormat='autodoc' -C OutputDirectory='.' -C
                    UseExistingComments=1
```

This command will produce a soardoc.soar file in the current directory with stub documentation blocks for all of your problem-spaces and operators. After this, run soardoc again normally and the datamap information should appear in the output.

I'm overriding configuration parameters on the command line with –C and SoarDoc complains?

If you're running SoarDoc on Linux, overridden config parameters must be enclosed in double quotes. See the Overriding Configuration Parameters section for more information.