The jSMTLIB User Guide

David R. Cok
GrammaTech, Inc.

Version 0.9.7
March 18, 2013

The most recent version of this document is available at
`http://www.grammatech.com/resource/smt/jSMTLIBUserGuide.pdf`.

Contents

Version History

2011-03-05	Version 0.6	Material added on adapters for non-compliant solvers
2011-01-02	Version 0.4	Relatively complete, matching feature-complete version 0.4 of the software
2010-12-31	Version 0.1	Initial draft

Note

This document almost certainly contains errors, as does the software it discusses. If you notice errors, please bring them to the author's attention and they will be corrected in a future edition. The software is accompanied by this disclaimer:

SOFTWARE DISCLAIMER OF WARRANTY/LIMITATION ON LIABILITY & REMEDIES. Author does not warrant that the SOFTWARE will meet your requirements, that operation of the SOFTWARE will be uninterrupted or error-free, or that all SOFTWARE errors will be corrected. THERE IS NO WARRANTY FOR THE SOFTWARE, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING, THE AUTHOR AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS AND WITH ALL FAULTS," WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS, IMPLIED OR STATUTORY (IF ANY), INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE SOFTWARE. IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING, WILL AUTHOR AND/OR ANY OTHER PARTY THAT PROVIDES THE SOFTWARE AS PERMITTED HEREIN, BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY GENERAL, SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING, BUT NOT LIMITED TO, ANY ECONOMIC LOSS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE OR ANY DATA SUPPLIED THEREWITH OR ARISING FROM ANY BREACH OF THIS AGREEMENT OR ANY OBLIGATIONS UNDER THIS AGREEMENT OR THE LICENSE GRANTED, EVEN IF AUTHOR OR ANYONE ELSE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, EVEN IN THE EVENT OF FAULT, TORT (INCLUDING NEGLIGENCE), MISREPRESENTATION, OR STRICT LIABILITY, OR FOR ANY CLAIM BY ANY OTHER PARTY.

If the disclaimer of warranty and limitation of liability provided herein cannot be given local legal effect according to their terms, reviewing courts shall apply the controlling law that most closely approximates an absolute waiver of all civil liability in connection with the SOFTWARE.

Chapter 1

Introduction

1.1 SMT solvers, the SMT-LIB language, and jSMTLIB

This document is a User Guide for the jSMTLIB software package. That package can be used as an application, as a network service, as a Java API, and as an Eclipse plug-in. It is useful as a parser and type-checker for SMT-LIB scripts, and as a translator of SMT-LIB scripts to the input languages of non-SMT-LIB-conforming SMT solvers. There is also a validation test suite that checks if an SMT solver conforms to SMT-LIB v.2.

The SMT-LIB language is described on the SMT-LIB website at `http://www.smt-lib.org` and in the SMT-LIB tutorial at `http://www.grammatech.com/resources/smt/SMTLIBTutorial.pdf`. It is a standard language for interacting with SMT solvers, which are software programs that solve logical constraint problems and are particularly appropriate for reasoning about software programs. The SMT-LIB language has been instrumental in the SMTCOMP, an annual public competition among various groups implementing SMT solvers; that competition has spurred significantly improved performance and capability in SMT solvers.

The jSMTLIB software package was produced and is maintained in conjunction with the SMTLIB Tutorial as an aid to users new to SMT-LIB and the use of SMT solvers. This User Guide supports the jSMTLIB software.

1.2 Mechanics

The document employs a few definitions and typographical conventions; these are described here.

A *conforming* SMT solver is one whose behavior obeys the SMT-LIB v.2 standard. Such a solver may do more than the standard requires, but not less. For example, it may define more options

or more commands, or it may be less restrictive in interpreting commands. However, any legal SMT-LIB v2 input must be accepted without complaint and yield the defined response.

This document is only concerned with SMT-LIB version 2; references to SMT-LIB are simply an abbreviated reference to version 2 of the language.

Verbatim characters. Text written using a monospaced font, e.g. `font`, represents character sequences that are to be interpreted verbatim. Typically, they are SMT-LIB input or output or fragments of SMT-LIB commands or shell commands in a computer OS.

Semantic categories. Character sequences such as <*path*> or <*int*> (using italics and enclosed in angle brackets) denote various semantic categories. For example, <*path*> typically denotes a file-system path to a given file or directory. These are not to be written verbatim, but are to be replaced by appropriate text.

Examples. The tutorial includes a number of examples, shown as text enclosed in a frame. For example,

```
> (set-logic QF_UF)
success
> (set-logic QF_UF)
(error "The logic has already been set")
```

Chapter 2

Download and Installation

There are a number of components related to the jSMTLIB software that can be separately downloaded; current versions of each of them are available from `http://www.grammatech.com/resources/smt`. Although they are distributed from GrammaTech's web site, this service is simply for the convenience of people interested in SMT solvers and SMT-LIB; these documents and software are not products of GrammaTech, and GrammaTech does not support or warranty them.

The following materials are available:

- an SMT-LIB tutorial:
  `http://www.grammatech.com/resources/smt/SMTLIBTutorial.pdf`
- the jSMTLIB software package:
  `http://www.grammatech.com/resources/smt/jSMTLIB.tar`
- this user guide to the jSMTLIB software and plug-in:
  `http://www.grammatech.com/resources/smt/jSMTLIBUserGuide.pdf`
- an Eclipse update site for an Eclipse plug-in for the jSMTLIB software:
  `http://www.grammatech.com/resources/smt/jSMTLIB-UpdateSite`

The jSMTLIB software package is downloaded as a .tar file from
`http://www.grammatech.com/resources/smt/jSMTLIB.tar`.
The tar file contains the following items:

- an executable Java jar file: jSMTLIB.jar
- a LICENSE file, granting permission to use the software for not-for-profit, academic uses
- a copy of this user guide, as a .pdf file
- `jsmtlib.properties-template`—you should edit this file for your own system configuration and save it as `jsmtlib.properties`, as described in section 3.4.7.

Installation simply consists of un-tarring the file into a directory of your choice, which will be designated in the sequel as `${SMTLIB}`, as in (on UNIX or MacOS X or Cygwin)

```
mkdir -p <directory>
cd <directory>
cp <download>/jSMTLIB.tar .
tar xf jSMTLIB.tar
```

or

```
SMTLIB=<directory>
mkdir -p ${SMTLIB}
cd ${SMTLIB}
cp <download>/jSMTLIB.tar .
tar xf jSMTLIB.tar
```

Here, <*directory*> represents the absolute path to the installation directory that you chose and <*download*> is the directory into which the tar file was downloaded.

On Windows, simply move the downloaded jSMTLIB.tar file into a new directory that you have made; then use 7-zip or a similar application to unpack the file into its consitutents.

Chapter 3

jSMTLIB as an SMT-LIB front-end application

3.1 Purpose of the application

As an application, jSMTLIB reads, checks and executes an SMT-LIB *script*. An SMT-LIB *script* is a sequence of SMT-LIB commands. The application reads each command in turn, parses and type-checks it, issues any error messages, and, if error-free, executes the command. The script can be provided in a file or directly input by the user at a input prompt.

If the jSMTLIB application is coupled with an *SMT solver*, then the execution of the script will normally result in the solving of a logical constraint problem posed by the commands in the script.

3.2 Running the application

This section describes running the jSMTLIB application from a command-line; it presumes

- you are using UNIX, MacOS, or Cygwin under Windows
- you have an installation of Java 1.6 as your default Java VM (available at as the `java` command)
- you are interacting with the OS through a bash shell (though translation to other shells is straightforward)

If you want to use a GUI to interact with the library, see section 7.

The command to run the jSMTLIB application consists of a command with typical options and files specified as arguments. Command options begin with double hyphens, with short-hand (one hyphen and one letter) abbreviations for some options.

Command. The Java application is run with this command:

```
java -jar ${SMTLIB}/jSMTLIB.jar <options> <files>
```

The permitted options are described in section 3.4.

Input. You may specify input scripts either by

- including the file system path to the file on the command line, or
- providing the input text directly through the process's standard input, or
- providing the script directly as text using the `-text` option described below (section 3.3.1), or
- providing the input through a network connection (section 3.3).

Input files on the command line. To process a specific file, put the file system path (<*path*>) to the file on the command-line. The contents of the file will be interpreted as an SMT-LIB command script.

```
java -jar ${SMTLIB}/jSMTLIB.jar <options> <path>
```

If more than one file is listed, the files are processed in order and individually (that is, not as one

concatenated file). For example, given an input file `p.smt2` containing

```
(set-logic )
(set-logic QF_UF)
(exit)
```

The command

```
java -jar ${SMTLIB}/jSMTLIB.jar p.smt2
```

produces the output

```
(set-logic )
           ^
(error "Expected a symbol here, not a )")
success
success
```

The first command has an intentional error, to demonstrate that in such a case, the tool echoes the offending text and issues an error message. The two succeeding commands each respond with a `success` response.

Input from standard input. If no files are listed (and no `-text` or `-port` options are present), then the tool reads its input from its standard input. This allows a user to directly interact with the tool. The tool will prompt for input with the `>` prompt. Typically one command is given per line, but in fact, input can be spread across lines — line breaks can be placed between any tokens of the input. If a prompt for new input is given while in the middle of a command, the prompt `...>` is printed.

To start the tool, issue this command (using the desired combination of options):

```
java -jar ${SMTLIB}/jSMTLIB.jar <options>
```

Here is an example, in which the bold font shows user input:

```
java -jar ${SMTLIB}/jSMTLIB.jar <options>
> (set-logic QF_UF)
success
> (quit)
   ^^^^
(error "Unknown command:  quit")
> (
...>exit
...>)
success
```

The first command, the set-logic command, receives a `success` response, indicating it executed successfully. The next command, `quit`, is not a valid command; the application responds by indicating the location of the error and an error message in SMT-LIB format. The last command, `(exit)`, is split across three lines to demonstrate the `...>` prompt that issued while in a command.

Sometimes in interactive mode, the parser has difficulty recovering from an error and insists (by showing the secondary prompt `...>`) that it is in the middle of parsing a command. To forcibly reset the parser to the beginning of a command, type a control-X.

In interactive mode, an end-of-input indication (a quick pseudo-exit command) can be input with either a control-D or control-Z character. Whether these are received and acted on by jSMTLIB may depend on your operating system and shell.

Redirected input. Input to standard-in can also be supplied by redirection from a file:

```
java -jar ${SMTLIB}/jSMTLIB.jar <options> < <path>
```

The application does not distinguish between input received in this manner from that received directly by typing. However, the error location information will not have text to point to, so this manner of supplying input is somewhat inconvenient. You might as well simply leave out the redirection character and have the input file be a command-line parameter:

```
java -jar ${SMTLIB}/jSMTLIB.jar <options> <path>
```

Exit values When the application terminates it provides an exit value to the operating system. That value is 0 if the application ran without errors and 1 if a command-line or script processing error occured.[1]

[1] The set of error codes may be changed or expanded in the future.

3.3 Running in client-server mode

The tool can also be run as a network service. In particular this allows the input to be sent from a conventional shell script. An SMT-LIB script does not have any control-flow constructs. For example, you cannot write an SMT-LIB script to do something different depending on whether a check for satisfiability of a set of assertions responds that the set is satisfiable or unsatisfiable. Rather than implementing such constructs in SMT-LIB, the client-server capability described here allows one to interact with the jSMTLIB application from a conventional shell.

This interaction operates as follows. One starts a service as a background process, listening on a given network port. Then jSMTLIB client processes send SMT-LIB commands (specified on the client process's command-line) to the service. The service processes the commands and sends responses back to the client. The client echoes the response to the user and a command shell can use the response in making control decisions or in forming subsequent input. In the current implementation, the client and service must be executing on the same machine.

To start the service, one selects an otherwise unused port number (say `5678` for these examples) and issues this command:

```
java -jar ${SMTLIB}/jSMTLIB.jar <options> -port 5678 &
```

The final & is so that the command runs in the background.

Then commands can be sent to this service from a client process using ordinary shell commands (in this case `bash` commands). Here the user input lines begin with the $ character (the shell prompt string in this case); the responses from the server are echoed back to the user, prefixed by `SMT:` .

```
$ CMD="java -cp ${SMTLIB}/jSMTLIB.jar org.smtlib.Driver -port 5678"
$ ${CMD} "(set-logic QF_UF)"
SMT: success
$ ${CMD} "(get-option :print-success)"
SMT: true
$ ${CMD} "(exit)"
SMT: success
```

There are a few items to note:

- The final exit command causes the service to terminate.
- Different clients communicating on the same port will send SMT-LIB commands to the same service, as if the SMT-LIB commands were simply interleaved.
- Different service processes must use different port numbers.
- The SMT-LIB text that is given on the shell command-line is typically enclosed in single- or double-quotes to avoid interpretation by the shell command-line processor.

- The client commands may include multiple SMT-LIB commands in the same command-line argument; however a command-line argument may not contain partial SMT-LIB commands.
- The client commands may have multiple command-line arguments — each is treated as SMT-LIB text to be sent to the service in turn.
- You have the full resources of the shell script language you are using to create the command-line arguments (that are the SMT-LIB commands). This allows reacting to errors or to the satisfiability or unsatisfiability results produced by the solver.
- return codes

The Driver program has these command-line options:

- -h or -help : print help information
- -v or –verbose : print diagnostic information
- -q or –nosuccess : do not print `success` responses
- -p <*int*> or –port <*int*> : the (required) port number on which to communicate

It returns these integer exit values:

- 0: if the response was `success`
- 2: if the response was `sat` to a check-sat command
- 3: if the response was `unsat` to a check-sat command
- 4: if the response was `unknown` to a check-sat command
- 5: if the response was some other non-error response
- 10: an SMT-LIB error
- 11: a command-line error
- 12: an unexpected exception

3.3.1 Input using quoted text on the command-line

One additional method of supplying a command script to the application is using the `-text` option. The `-text` option requires an argument; that argument is used as the text of an SMT-LIB script, directly placed on the command-line. An example is given in the discussion of the `-text` option below.

Long form	Short form	Description
–help	-h	usage information
–version		version information
–text <*string*>		text to use as the SMT-LIB script
–port <*int*>		network port on which to listen
–solver <*name*>	-s <*name*>	which solver to use (default is type-checking only)
–exec <*path*>	-e <*path*>	path to the solver executable
–logics <*directory*>	-L <*directory*>	directory containing logic definitions (default is to use internal definitions)
–out <*file*>		path to file to which to append regular output
–diag <*file*>		path to file to which to append diagnostic output
–nosuccess	-q	turn off echoing of `success` response
–noshow		do not show error locations, just error messages
–echo		enable echoing of commands to the output as they are processed
–verbose <*int*>	-v <*int*>	set the verbosity level
–relax		enable some extensions to SMT-LIB

Table 3.1: Summary of command-line options

3.4 Command options

The command-line tool has a number of command-line options that control its behavior, as summarized in Table 3.1. Each has a full-name, beginning with two hyphen characters; some of the options also have an equivalent short form, beginning with one hyphen. Some of the options have a required argument; some take no argument.

3.4.1 General information

–help or **-h** This option prints (to the standard output) a lengthy message describing the use of the command and all of the options and then exits the application.

–version This option prints a brief message showing the current version of the software and then exits the application.

3.4.2 Source of input

The application can receive input through the standard input or command-line files, through a text string on the command-line, or through a network port. If `-text` is specified, it is used; otherwise

if `-port` is specified, it is used; otherwise if one or more files are listed on the command-line, they are used; otherwise the standard input is prompted for input.

–text <*string*> The argument is interpreted as text and processed as a script of SMT-LIB commands. For example,

```
java -jar ${SMTLIB}/jSMTLIB.jar -text '(set-logic QF_UF)(exit)'
```

produces

```
success
success
```

and

```
java -jar ${SMTLIB}/jSMTLIB.jar -text '(set-logic QF_UF)(ex)'
```

produces

```
success
(set-logic QF_UF)(et)
                  ^^
(error "Unknown command:  et")
```

It is not allowed to also specify the `-port` option; if files are listed on the command-line they are ignored during command-line use.

–port <*int*> This option directs the application to listen on the given network port for input to process. The application will accept input and respond to the port with its output until it receives and responds to a `(exit)` command, at which point it will terminate. This mode of operation is described in section 3.3.

3.4.3 Specifying the back-end solver

The jSMTLIB application is simply a front-end. It can parse and type-check SMT-LIB scripts, but it does not do any actual constraint solving. However, it does have adapters that enable it to translate strict SMT-LIB input into the input language of other non-SMT-LIB-conforming solvers. The combination then becomes an SMT-LIB-conforming solver. If you have a SMT-LIB-conforming solver, you can use it directly to process SMT-LIB scripts; even then the jSMTLIB application may be useful as a light-weight checker for SMT-LIB scripts. If you have a non-SMT-LIB-conforming solver, an appropriate adapter (cf. section 4) may make it possible for the solver to be used with SMT-LIB scripts.

To use a solver through an adapter you need the name <*name*> of the solver (which corresponds to the name of Java class implementing the adapter) and the absolute file-system path to the

Solver	Available from
Simplify 1.5.4	`https://mobius.ucd.ie/repos/src/mobius.esc/escjava/trunk/ESCTools/Escjava/release/master/bin/`
Yices 1.0.28	`http://yices.csl.sri.com/`
CVC3 2.2	`http://cs.nyu.edu/acsys/cvc3/`
Z3 2.11	`http://research.microsoft.com/en-us/um/redmond/projects/z3/`

Table 3.2: Solvers for which adapters are available

executable <*exec-path*> on your system. If no solver is specified, the application only does parsing and type-checking of the SMT-LIB script.

Table 3.2 lists the solvers for which adapters are currently available. The solvers themselves are not supplied with jSMTLIB. (They may require licenses from their authors.)

–solver <*name*> or **-s** <*name*> This option specifies the name of the solver to use. If no solver is specified, the application does just parsing and typechecking. The currently supported solvers are `simplify`, `yices`, `cvc`, and `z3`.

–exec <*exec-path*> or **-e** <*exec-path*> This option specifies the file-system path to the executable file for the solver on your system. If the value is not specified on the command-line, it is taken from the `jsmtlib.properties` file (cf. section 3.4.7).

–logics <*directory*> or **-L** <*directory*> By default, the tool uses an internal set of logic and theory definitions. This option directs the tool to look in the given directory for such definitions. A (`set-logic` <*name*>) command will fail if the tool does not find a <*name*>`.smt2` file in the given directory (or by default, in the tool's internal files). Some aspects of logics and theories have to be hard-coded into the interpreters, so adding a new theory or logic is not as simple as adding a new definition file, though that is a required step.

–abort This option sets the tool behavior to immediate exit (section 3.4.8). In `immediate-exit` mode, the tool will abort processing a script (in batch mode) or the line of text (in interactive mode) upon encountering the first error. Without this option, the behavior is `continued-execution`, in which the tool does its best to recover from the error and continue processing the input text. Type-checking mode always continues after errors, despite this option.

3.4.4 Control of output location

An SMT-LIB tool sends its error and successful command responses to what is called the regular-output-channel; it sends any diagnostic information to a diagnostic-output-channel. By default these are the standard output and standard error streams of the process. These can be redirected by SMT-LIB commands to a user-specified file. They can also be set initially to user-specified files using these two options.

–out <*filename*> This option sets the regular-output-channel to append to the given file, creating the file if necessary.

–diag <*filename*> This option sets the diagnostic-output-channel to append to the given file, creating the file if necessary.

Neither option creates the containing directories.

3.4.5 Filtering of output content

–nosuccess or **-q** This option omits the printing of `success` in response to the successful execution of commands that give that response. Any error messages or non-error responses that are not `success` are still printed. This option is equivalent to sending the command `(set-option :print-success false)` at the beginning of the SMT-LIB script.

–verbose <*int*> or **-v** <*int*> This option enables diagnostic printing from the application and any back-end solver. The option requires a non-negative integer argument. The default value is 0 and corresponds to no diagnostic printing at all. Generally, increasing values of the argument should enable increasing amounts of input, but that behavior is solver dependent. The application itself does not distinguish between different positive values. The option is equivalent to setting the `:verbosity` option in SMT-LIB with the given argument (that is, sending the command `(set-option :verbosity` <*int*>`)`.

–echo This option enables the echoing of commands just before they are executed. When a file is listed on the command-line, the responses to each command are sent to the output, but ordinarily the original command is not. This can make it difficult to know which commands in a lengthy script have been executed. Echoing the input commands provides an indication of progress. When commands are typed directly to standard input, such echoing is not necessary because the input is evident. If a command is malformed and does not parse successfully, it will not be echoed to the output.

–noshow This option disables the showing of error location and only outputs the error message itself. This can make the output less verbose and also makes the output strictly conform to the SMT-LIB standard, but generally also makes it harder to locate the error in the text. [TBD - may change the implementation to put the location information in the text of the message when `-noshow` is enabled.]

3.4.6 Extensions

–relax This option relaxes strict conformity to the SMT-LIB standard and enables some extensions. The extensions enabled currently are these:

- Allowing some additional commands: `what`, `exec`
- Allowing SMT-LIB command names as symbols (rather than as reserved words)

3.4.7 Properties file

The application also uses a properties file to determine some default settings that may be system dependent. The file is named `jsmtlib.properties`. The applications looks for and loads properties from files named `jsmtlib.properties` from the following locations, in order, with values set in later files overriding those in earlier files:

- on the application's classpath
- in the directory in which the jSMTLIB.jar file resides
- in the user's home directory (as given by the Java property `user.home`)
- in the current working directory

The distribution contains a template for the properties file, named `jsmtlib.properties-template`. That file contains a list and description of all of the properties that have been defined. You should make a copy of the template file, rename it as `jsmtlib.properties`, edit it to reflect your own system configuration, and store it in one of the locations listed above.

The most useful values to add to the properties files are the file system locations of solver executables. Once you have installed the solvers on your system, their locations are generally stable; putting those locations in the properties file avoids having to put them on the command-line.

3.4.8 Errors, error behavior, and error recovery

Errors can occur because of invalid tokens (lexer errors), incorrectly constructed commands or expressions (parsing errors), incorrect use of sorts (type errors), or run-time errors in the course of

executing the SMT solver (run-time errors). The application does its best to recover from errors and to proceed. In type-checking, this enables finding as many errors in one pass as possible.

- Type-checking errors are always recoverable. The command in which the error occurred is skipped, and processing can continue with the next command.
- Lexer and parsing errors are generally recoverable; the parser may become confused if there are mismatched parentheses or unclosed strings or quoted symbols.
- Run-time errors are generally not recoverable.

Errors are reported with standard SMT-LIB error responses. The SMT-LIB standard only accommodates a single error response per command, so in a text-based application, only one of perhaps multiple errors will be reported. (The Eclipse GUI does not have this limitation). By default, error messages are accompanied with text showing the error location; the `-noshow` option allows the user to request that the error location be suppressed, showing only the message.

The SMT-LIB standard prescribes two modes of responding to errors: `continued-execution` and `immediate-exit`. Continued-execution is the default behavior; immediate-exit behavior can be chosen with the `-abort` command-line option. When the application is operating as a type-checking tool (and not doing any back-end solving), it always operates in continued-execution mode.

continued-execution: In this mode, a command containing an error has no effect on the solver state; the application does its best to continue with any following commands.

immediate-exit: In this mode, execution of a command script halts immediately upon the first error. This mode makes the most sense for batch processing of an entire file, because an error in the script generally will require correction and reprocessing— processing a script with errors is often a waste of time. In interactive mode, however, the user has the opportunity to enter a corrected command, if an error is inadvertently entered. Thus in interactive mode, `immediate-exit` is interpreted as aborting the processing of any text on the same line as the error.

Chapter 4

SMT adapters and solvers

The jSMTLIB library includes several adapters that translate SMTLIB format commands into a format read by a non-SMTLIB solver. Not all features of SMT-LIBv2 can be translated, depending on the capabilities of the target solver.

Commands are directed to a backend solver using the `-solver` and `-exec` command-line options; one can also use properties specified in the properties file. The value of the `-solver` option is a string giving the name of the solver to use; this corresponds to an internal class that does the format translation. The value of the `-exec` option is a string giving the absolute file-system location of the executable that constitutes the solver. For example, to run the yices solver on my system, I can use the command
-jar jmstlib.jar –solver yices –exec C:/cyginw/home/dcok/mybin/yices.exe

It is more convenient to put the executable paths in the jsmtlib.properties file. The property that specifies the executable for a solver named *solver* is org.smtlib.solver_*solver*. For example, my properties file contains the following lines, giving locations for the yices, cvc, z3 and simplify solvers:

```
org.smtlib.solver_yices=C:/cygwin/home/dcok/apps/yices-1.0.28/bin/yices.exe
org.smtlib.solver_cvc=C:/cygwin/home/dcok/apps/cvc3-2.2-win32-opt.exe
org.smtlib.solver_z3="C:/Program Files (x86)/Microsoft Research/Z3-2.11/bin/z3.exe"
org.smtlib.solver_simplify=C:/cygwin/home/dcok/mybin/simplify.exe
```

With these in place, the command-line `-exec` option is not needed.

Four adapters are currently implemented, as described in the following sections. All adapters, and the jSMTLIB checker itself, have these current limitations:

- attributed expressions within let constructs are not properly handled

- there is no check that expressions conform to the specified logic
- the error-behavior is not consistently implemented or reported

4.1 Simplify

The `simplify` adapter is designed to work with Windows version 1.5.4 of the Simplify solver, on either Windows or Cygwin. The adapter currently has the following limitations as of the date of this user guide:

- the bit-vector theory is not supported, including using binary or hex literals and bit-vector functions
- the let construct is not implemented
- decimal literals are not implemented
- Boolean values and variables are not allowed as terms or as variables in quantifiers
- the get-proof, get-unsat-core, get-value, and get-assignment commands and the corresponding produce- options are not supported

4.2 Yices

The yices prover, available from SRI at `http://yices.csl.sri.com/`, does not yet support SMT-LIB. The adapter included with jSMTLIB provides an interface to the Yices version 1.0.28 on Windows or Cygwin. Use the statically-linked download for Windows.

The Yices adapter has these limitations:

- parameterized sorts, other than the built in Array sort, are not permitted
- attributed expressions within let or quantified expressions are not implemented
- decimal literals are not implemented
- the bit-vector functions bvshl, bvlshr, bvudiv, bvurem are not implemented
- the get-proof, get-unsat-core, get-value, and get-assignment commands and the corresponding produce- options are not supported

4.3 cvc

The CVC3 prover is available from `http://www.cs.nyu.edu/acsys/cvc3/`. The adapter included with jSMTLIB provides an interface to the CVC3 version 2.2 on Windows or Cygwin.

The cvc adapter has these limitations:

- parameterized sorts, other than the built in Array sort, are not implemented
- attributed expressions within let or quantified expressions are not fully implemented
- decimal literals are not implemented
- the bit-vector functions bvshl, bvlshr, bvudiv, bvurem are not implemented
- functions may not be defined with Bool arguments
- the if-then-else construct may not be used as a term
- push and pop do not affect definitions and declarations of sorts and functions
- the get-proof, get-unsat-core, get-value, and get-assignment commands and the corresponding produce- options are not supported

4.4 z3

The Z3 prover is available from `http://research.microsoft.com/en-us/um/redmond/projects/z3/index.html` (and has license restrictions). The adapter included with jSMTLIB provides an interface to the Z3 version 2.11 on Windows or Cygwin.

The Z3 adapter has these limitations:

- it recognizes the Array sort and select and store functions regardless of the logic chosen
- it does not parse SMT-LIBv2 binary and hex literals
- declarations and definitions and the get-assertions command are not scoped with push and pop
- it does not report errors when a sort name is redeclared
- attributed expressions do not result in declarations

Chapter 5

jSMTLIB as an API

The jSMTLIB software also serves as an API, enabling the construction of SMT-LIB expressions, sorts, and commands directly from Java programs. This section provides an overall survey of the API; the details of individual API calls can be found in the appropriate javadoc documentation, located at TBD. Two general points a worth noting:

- Most of the public API is contained in the `org.smtlib` package. There may be methods and classes in other packages that have public visibility, but unless explicitly indicated, the public visibility is used simply to enable use of the classes within the various packages of the jSMTLIB library; public visibility does not necessarily imply the class or method is part of the public (stable) API.
- In the descriptions that follow, classes and interfaces are contained in the `org.smtlib` package, unless otherwise indicated.
- The public API is nearly all expressed in Java interfaces. All new object construction uses factories. The combination of these design patterns allows easy extension of the library: the programmer (a) writes new classes that implement the appropriate interfaces, (b) writes a factory that generates the new objects, conforming to the factory interface, and (c) registers the factory in the library so that it is used when needed to allocate new objects.
- Although there is an API to construct SMT-LIB expressions, the interfaces to solvers are still textual, with the solvers running as separate processes.
- This chapter contains some information about extending the library, but more detail and code examples can be found in the following chapter.

5.1 SMT Contexts and Configurations

All jSMTLIB operations occur with respect to a context. A context consists of a configuration and the current state. Some than one context can be active at once, but they are logically separate. A context implements the `ISMT` interface.

An SMT configuration contains a record of the properties to be applied to operations (e.g. the verboseness of output or which solver to use by default) and which factories to use for operations within the context. A configuration does not retain any state; thus configurations can be copied from one context and applied to another. A configuration implements the `IConfiguration` interface.

A context also contains some state (not in the configuration). For example, as expressions are created and typechecked, the context maintains the symbol table holding the sorts and functions that have been defined so far. The state also receives counterexample or proof information from solvers.

The `org.smtlib.Root` class contains static methods that allow setting defaults and creating new contexts. A context may be created with the call

```
ISMT context = Root.newSMT();
```

An initial context is created with a default `IConfiguration`; both are created using the default factories and properties stored in the `Root` class.

A context may also be created from an existing context:

```
ISMT context = Root.newSMT();
ISMT context2 = context.newSMT();
```

The configuration of the new context is a clone of the configuration of the old context. The state is not copied; the new context has an initialized (empty) state.

The (reference to the) configuration of a context can be retrieved and set with these calls:

```
IConfiguration config = context.config();
...
IConfiguration oldConfig = context.setConfig(newConfig);
```

The `setConfig` call returns a reference to the previous configuration. A copy of a configuration can be obtained with the standard `clone()` method:

```
IConfiguration copy = config.clone();
```

5.2 Logging information

All error messages and informational output from the jSMTLIB library is emitted through calls to a logging object. The logging object implements the interface `ILog`. The `ILog` object used is the object registered in the IConfiguration in current use. A default `ILog` object is implemented by the jSMTLIB library unless replaced by a user-supplied `ILog` object.

The `ILog` interface permits listeners to subscribe to it. The listeners implement the `ILog.IListener` interface and are registered with the ILog object whose messages they wish to hear. If there are no listeners registered, then by default the standard `ILog` object will send output to `System.out` and `System.err`, but user-defined logging objects may implement other behavior.

The `ILog` interface divides the output from the jSMTLIB library into three kinds of output:

- regular output (`logOut` methods): these methods receive standard SMT-LIB output
- error output (`logErr` methods): these methods are called with any error messages
- diagnostic output (`logDiag` methods): these methods are called with general diagnostic or progress information; in very quiet (non-verbose) operation, none of this kind of output would be created

Recall that the SMT-LIB standard specifies two output channels: regular output (which includes normal command responses and error responses) and diagnostic output. The format of regular output is prescribed by the standard; the default ILog object sends the output of logOut and logErr methods to the regular output channel. The format of diagnostic output is completely unspecified; the default ILog objecct sends the output of logDiag methods to the diagnostic channel. Additional configuration options determine where the regular output channel and the diagnostic output channel go: to a file, to `System.out`, or to `System.err`.

5.3 Properties

Properties control the general behavior of the library. There are separate sets of properties for each configuration. A configuration's properties are initialized from default settings and ccan be modified programmatically. Cloning a configuration creates a new copy of the properties.

Properties in jSMTLIB are modeled after properties as otherwise used in Java (i.e. in the `java.util.Properties` class). They are `String` values with `String` keys. Some properties expect that the property value has a certain form. For example, some expect a fully-qualified class name, others expect a Boolean value.

Key	Type of value	Default value	Description
`defaultSolver`	String	`test`	the name of the solver to use by default
`verbose`	Integer	0	the verbosity level
`solver.<name>`	Path	-	executable for a solver adapter

Table 5.1: Summary of recognized properties

Properties are initialized from a typical Java properties file. The library initialization code looks for a properties file named `jsmtlib.properties` in TBD... The properties read from these files become the default properties initially set in `Root`.

Table 5.1 lists the properties recognized by the current version of jSMTLIB; expanded descriptions are provided for some of the properties below. All key values have a prefix of `org.smtlib.` (including the final period). For example, you can set the verbosity level, by setting the value of the `org.smtlib.verbose` property to a positive integer (0 turns off all diagnostic output).

5.3.1 Description of properties

defaultSolver . The value gives the name of the solver adapter to use when a SMT-LIB script is executed. Currently defined values are `z3`, `simplify`, `cvc`, `yices`, and `test`. The value `test` is the default and indicates to do only type-checking.

verbose . The value is a non-negative integer. The value `0` indicates complete quiet - no informational output at all. Larger values give increasing amounts of diagnostic or informational or debugging output (on a not clearly defined scale).

TBD - more needed: destination of output channels; executables; factories

5.4 Factories

5.5 Creating sorts and expressions

Expression trees are created programmatically by building up the tree node by node. All nodes representing a given syntactical category are objects of a class that implements a given interface. For example, a SMT-LIB *numeral* implements the `IExpr.INumeral` interface; a SMT-LIB *symbol* implements the `IExpr.ISymbol` interface; a quantified (i.e. `forall` pr `exists`) expression implements the `IExpr.IQuantifiedExpression` (TBD-check) interface.

Nodes are produced using a given factory. Different factories may produce nodes with different internal implementations, but the nodes always implement the interfaces as decribed above and can always be handled using interface-typed references.

The jSMTLIB library supplies two expression factory classes. One factory, instances of TBD, produces AST nodes containing no additional information. The factory is most appropriate for programmatically produced ASTs.

A second factory class, TBD, is used by the built-in parser. It produces nodes that contain position information. That is, each node contains information telling the range of source file characters corresponding to the expression. Having such information available enables error messages or GUIs to relate the AST to displayed positions in the source text.

Note that an AST may have shared sub-trees — that is, it may be a DAG rather than a tree.

TBD - equality of sorts and expressions

TBD - cloning of trees

TBD - issues for symbols

TBD - example

5.6 Creating commands and scripts

5.7 Visitors

TBD - tree visitors, translators, copiers

5.8 Printing

A printer converts an internal abstract syntax tree into concrete syntax. There is a default printer that writes an AST to SMT-LIBv2 standard concrete syntax. Users can write other printers to generate other external representations of an AST. A printer is easily implemented as a visitor class that walks an AST and knows how to print each kind of node that might be encountered.

- A printer can be set programmatically for a configuration using ... TBD
- The default printer is an instance of the `org.smtlib.` ... TBD class

- The default printer can be set to something else using the property ... TBD
- A configuration's default printer is used in many places. For example, error messages that refer to expressions will render the expression using the default printer for the current configuration.

TBD - need examples; interface? works from interfaces, so nodes from any factory should work

5.9 Parsing

A parser converts external concrete syntax into an internal AST.

TBD - sources of input, interface, textual locations, uses a factory to produce nodes

5.10 Typechecking and symbol tables

5.11 S-expressions

5.12 Solvers

Information will be provided in a future edition.

Chapter 6

Programmatically extending the application or the API

Information will be provided in a future edition.

Chapter 7

The Eclipse plug-in for jSMTLIB

An Eclipse plug-in that encapsulates the jSMTLIB application is available from the update site: `http://www.grammatech.com/resources/smt/jSMTLIB-UpdateSite`.

The plug-in is an IDE for editing .smt2 scripts, for executing solvers on those scripts, and for reviewing the results.

Note that the file-system paths to desired SMT solvers must be set in the SMT Preferences page before the solvers can be successfully invoked by the plug-in. The Preferences page is described in section 7.6 below.

7.1 SMT Editor

The `.smt2` file suffix is associated with an SMT Editor. The editor presumes the content of the `.smt2` file is a SMTLIBv2 command script. The SMT editor provides

- syntax coloring for the lexical elements of the SMTLIBv2 file. For example parentheses are magenta, keywords are blue, and variables are green.
- immediate syntax and type-checking of the contents of a file.

Note that an Eclipse user can change the file associations on the `General » Editors » File Associations` preference page. So the user can associate files with a different suffix (or other name pattern) with the SMT Editor.

Files that are identified as SMT Editor files are designated with a special SMT icon. That icon is a blue disk containing a white S.

Some features planned for the future but not currently implemented are

- showing and navigating between matching parentheses;
- navigating to the declaration of a variable;
- showing all uses of a given declaration;
- refactoring actions (e.g., renaming all uses of a variable);
- user setting of the colors for syntax coloring;
- being able to set the fonts as well as colors in syntax coloring.

7.2 Console

The plug-in provides a new type of Console, the SMT Console. This console receives all output from solvers as they are executed. Error messages may appear in the console or in pop-up dialog boxes or both. [TODO- what about the Eclipse error log]

7.3 Problem markers

The checking of a file is automatic and happens as the file is edited. Syntax and type errors in a file are identified by Problem Markers and by text annotations.

Problem Markers are shown in either or both the vertical ruler on the left side of an editor window and the overview ruler on the right side of an editor window. In the vertical ruler problems are shown as red disks with a superimposed white S. Each problem is also listed in the Problems View. A user can navigate among problems and from a line in the Problems View to the corresponding line in the text just as for other Eclipse Problems.

Text annotations highlight the particular text within a line that is associated with a Problem marker. Text annotations can be either squiggly underscores or highlighted text. The user can make this choice in the Eclipse Preferences under `General » Editors » Text Editors » Annotations`. Using that preferences page, the user can set

- whether indications appear in the vertical (left) ruler,
- whether problem indications appear in the overview (right) ruler,
- whether problem indications appear in the text,

- whether text annotations are squiggly underscores or highlights, and
- the color of the underscores or highlights.

7.4 Menu items

User actions can be instigated in four ways:

- A new menubar item, named `SMT`, provides a number of menu items with corresponding actions
- A context menu provides most of the same actions. It is available by right-clicking on a file or directory in the it Package Explorer or *Navigator* views. There is a top-level menu item named `SMT`, with a number of submenu items.
- A toolbar item, identified by the SMT plug-in's blue disk icon, instigates the *Run default solver* action.
- Actions can be instigated by keyboard combinations, if those are defined as described in the section 7.5.

The plug-in implements the following actions:

- `Run default solver` : Runs the solver selected on the Preferences page on the currently selected files (see the description about running solvers below)
- `Delete SMT markers` : Normally old SMT problem markers are deleted when a file is rechecked and updated markers created. However, sometimes it is desirable simply to delete the current set of markers. A bug can also prevent current markers from being properly deleted. This action will delete all markers from currently selected files or editors. Markers will be recreated as the file is edited or by invoking the `check` command.
- `View Logic` : This action will bring up the contents of the selected SMT Logic or Theory in a separate text editor. The desired Logic or Theory is designated by selecting its name in any existing text or SMT file (or by typing its name and then selecting the typed text). For example, most command scripts begin with a set-logic command, such as `(set-logic QF_UF)`; the name of the logic in the set-logic command can be textually selected and then the View Logic operation invoked. The displayed Logic or Theory cannot be edited by through the text editor in which it is shown. (TODO - is this future functionality?)

- `Get Value` : If a solver has just been run on an SMT file and returned a result of `sat`, then there is a satisfying assignment for the variables in the command script. If the underlying solver supports the optional `:produce-models` capability, one can explore the values of the satisfying assignment through the GUI. To do this, select an expression in the editor and then initiate the `Get Value` commmand. The expression selected may be a single variable (e.g., `xy`), a parenthesized expression (e.g., `(+ x y)`, or a sequence of expressions (e.g., `x y`).
- `All`: Runs all configured solvers on the selected files
- `check` : Performs syntax and type-checking on the selected files or editors. This is done automatically while editing, but can also be initiated manually using this command.
- a specific solver: Runs the specificly chosen solver, each of which has an individual menu item identified by the name of the solver.

The `All`, `check`, `Run default solver`, and the specific solver items all run the solvers as follows. The solvers act on the set of selected SMT files. You can select one or more (using shift-select or control-select) files in the Package Explorer View; if a folder (directory) of files is selected, then all SMT files within that folder (recursively) are implicitly selected. If no files are selected, the solver acts on the contents of the currently selected editor. Note that the solver acts on the current edited but not necesarily saved contents of the editor. If no editor and no files are selected, an error dialog appears.

The plug-in initiates a single background job to execute the requested action. It iterates over each selected file, and for each file, applies the selected solver (or all the solvers in turn). The results of each solver attempt are recorded in the SMT Console View. Progress is also shown in a Progress View. However, canceling the progress view does not cancel the solver invocation currently underway, just as yet unstarted invocations. To cancel the solver currently running, one currently must kill the process using, for example, the Windows Task Manager.

Future functionality: setting solver timeouts; more convenient aborting of solvers; more convenient display of a set of solver results.

7.5 Commands and keyboard bindings

Each of the actions available through menu items is an Eclipse command and can be associated with a keyboard key sequence, so that the action can be invoked with the keyboard, without using the mouse. Eclipse commands are listed in the `General » Keys` page of the Eclipse preferences. The SMT commands are all under the `SMT` category, so if the sorting of the commands is by category all the SMT commands will be together.

Just as for other Eclipse commands, a user-chosen key binding can be associated with any given command.

7.6 Preferences

The SMT Plug-in implements a conventional Eclipse Preferences page, named `SMT`. The preferences apply to all projects. There is as yet no capability to set project-specific preferences. The SMT Preferences are on two tabs: a general settings tab and a solver executable paths tab.

General settings page:

- `verbose plugin?` – enabling this option causes more informational and debug messages about the operation of the plugin to be written to the console
- `verbose SMT app?` – enabling this option causes more informational and debug messages about the operation of the underlying SMT application to be written to the console. This corresponds to the `-verbose` or `-v` command-line flag.
- `solver verbosity` – the value to be set as the `:verbosity` SMTLIBv2 option in each SMT solver
- `print success?` – if enabled, the `success` is printed as appropriate as the tool executes each step of a command-script. This corresponds to enabling or disabling the standard SMTLIBv2 `:print-success` option
- `abort on error?` – (TODO - to be implemented)
- `echo batch commands?` – if enabled, then the GUI echos each script command to the console as it is processed and executed
- `relax conformity?` – if enabled, the tool allows some extensions to the standard SMTLIBv2, as described in TODO. If disabled, strict SMTLIBv2 conformance is required.
- `Default solver to use` – Sets which solver is used by the toolbar button and the "Run default solver" menu item.
- `implicit logic` – TODO - to be implemented
- `Directory containing logic files` – SMTLIBv2 uses a set of logic and theory definition files to determine which logics are allowed and their definitions. The installation contains a set of built-in logics. However, the user can set a directory path to an alternate set of logics if desired.

Solver paths page: The jSMTLIB tool is an insterface and adapter to various SMT solvers, but does not actually include those solvers. Hence the tool needs to be told where to find the desired solvers. This preference page allows the user to set the paths to each solver. It is the equivalent of corresponding entries in the jsmtlib.properties file and the `-exec` command-line option. The page lists the available solvers and provides a text field in which one can enter the path to the executable for that solver; alternately, one can use the `Browse` button to navigate to and select the executable using a conventional file browser.

Future functionality: the ability to add new solver entries to this page.

7.7 Internationalization

Eclipse provides a mechanism to internationalize the user-visible messages and labels produced by a plug-in. The SMT plug-in is not internationalizing warning and error messages. However, GUI labels (e.g., menu item labels, command descriptions) are in the process of being internationalized.

Alternate text can be provided for GUI strings by replacing or editing the `OSGI-INF/I10n/bundle.properties` file.

An index will be added in a future edition of the user guide.