SICStus Prolog Release Notes

by the Intelligent Systems Laboratory

Swedish Institute of Computer Science
PO Box 1263
SE-164 29 Kista, Sweden

October 1999

Swedish Institute of Computer Science
sicstus-request@sics.se
http://www.sics.se/sicstus/

Swedish Institute of Computer Science
PO Box 1263
SE-164 29 Kista, Sweden

1 Release notes and installation guide for UNIX

This chapter assumes that the environment variable `PATH` includes `<prefix>/bin`, where `<prefix>` points to the SICStus installation directory. The installation directory is specified during installation, see Section 1.3 [UNIX installation], page 1. For example:

```
csh,tcsh> setenv PATH "/usr/local/bin:$path"
sh,bash,ksh> export PATH="/usr/local/bin:$path"
```

1.1 Setting SP_PATH under UNIX

The `SP_PATH` environment variable can be used to override the location of the SICStus Runtime Library. There are two situations where it is justified to use it.

1. The `--moveable` option has been given to `spld` when building the executable.
2. The executable is a runtime system, and `NULL` is passed in the third argument to `SP_initialize`.

The correct value of this variable in both cases is '`/usr/local/lib/sicstus-3.8`', assuming SICStus was installed in '`/usr/local`'.

1.2 The Crypt Utility

The SICStus binary distributions are encrypted with the `crypt` program. If you do not have `crypt` on your machine, you can download a public domain crypt utility available via anonymous FTP from

`ftp://ftp.sics.se/archive/sicstus3/crypt.tar.gz`

The enclosed README files describes how to compile it.

1.3 Installation

Most users will install SICStus from a binary distribution. These are available for all supported platforms. Information on how to download and unpack the binary distribution is sent by email when ordering SICStus.

Binary distributions are installed by executing a interactive installation script called `InstallSICStus`. Type

```
% InstallSICStus
```

and follow the instructions on the screen.

During the installation, you will be required to enter your site-name and license code. These are included in the download instructions.

The installation program does not only copy files to their destination, it also performs final link steps for some of the executables and for the library modules requiring third-party software support (currently `library(bdb)`, `library(tcltk)`, and `library(jasper)`). This is done in order to adapt to local variations in installation paths and versions.

Compiling SICStus from the sources requires a source code distribution, available on request for customers with maintenance contract. Contact `sicstus@sics.se` for more info.

Instructions for compiling and installing SICStus from the source code is available in the files `README` and `INSTALL` in the source code distribution.

1.4 Foreign language interface

1.4.1 How to customize splfr and spld

The utilities `splfr` and `spld` are implemented as Perl scripts and can be customized in order to adapt to local variations. *Do not attempt this unless you know what you are doing.* Customization is done by editing their common configuration file `spld.config`. Follow these instructions:

1. Locate the configuration file `spld.config`. It should be located in the same directory as `splfr` and `spld`.
2. Make a copy for `spld.config`, lets call it `hacked_spld.config`. Do not edit the original file.
3. The configuration file contains lines on the form `CFLAGS=-g -O2`. Edit these according to your needs. Do not add or remove any flags.
4. You may now use the modified `spld.config` together with `spld` or `splfr` like this
   ```
   % spld [...] --config=/path/to/hacked_spld.config
   ```
   Replace `/path/to` with the actual path to the hacked configuration file.

1.4.2 How to create dynamic linked foreign resources manually

To compile the glue code file and user code, use the compiler options assigned to `INCR_CFLAGS` by `./configure`. In addition also include `-DSPDLL`.

The object files are then linked into a dynamic linked foreign resource. For this you will normally use the linker whose name was assigned to `SHLD` by `./configure` and linker options assigned to `SHLDFLAGS`. The resource will consist of the file *ResourceName.Suffix* where *Suffix* is the value assigned to `SHSFX` by `./configure`. The defaults are

```
SHLD= ld
SHLDFLAGS= -shared
SHSFX= so
```

E.g. on Sparc/SunOS 5.X:

```
% cc -c -DSPDLL glue_code.c
```

```
% cc -c -DSPDLL mycode.c
% ld -shared glue_code.o mycode.o -o myresource.so
```

Libraries needed by the resource should normally also be included in the link command line.

1.4.3 Interfacing to C++

Functions in C++ files which should be called from Prolog must be enclosed like e.g:

```
extern "C" {
void myfun(long i)
{...};
};
```

To build a dynamic linked foreign resource with C++ code, you may (depending on platform) have to explicitly include certain libraries. E.g. on Sparc/SunOS 5.X using gcc:

```
% splfr .... -LD -L/usr/gnu/lib/gcc-lib/sparc-sun-solaris2.4/2.7.0 -lgcc
```

The library path is installation dependent, of course.

1.5 Platform specific UNIX notes

There are currently no platform specific notes.

This section contains some installation notes which are platform specific under UNIX.

- MacOS X Server The following libraries are not supported: `library(bdb)`, `library(db)`, `library(tcltk)`, `library(jasper)`.

1.6 Files that may be redistributed with runtime systems

When a runtime system is redistributed to third parties, only the following files may be included in the distribution. All filenames are relative to '`<prefix>/lib/sicstus-3.8`':

'`../*.{a,so,sl,dylib}`'

'`bin/sprt.sav`'

'`bin/jasper.jar`'

'`library/*.{tcl`'
,po}

'`library/*/*.{s.o,so,sl,dylib}`'

'`library/*/*.po`'

2 Release notes and installation guide for Windows

This chapter assumes that the environment variable `PATH` includes `%SP_PATH%\bin`, where `SP_PATH` points to the SICStus installation directory. For example:

```
C:\> set PATH=c:\Program Files\sicstus3\bin;%PATH%
```

You may also want to include the paths to Tcl/Tk (see Chapter 3 [Tcl/Tk notes], page 10), Java (see Section 4.1 [Getting Started], page 11), and Berkeley DB (see Chapter 6 [Berkeley DB notes], page 16).

2.1 Requirements

- Operating environment: Microsoft Windows 95, 98, NT 4.0.
- Processor: 386, 486, or Pentium-class
- Available user memory: 16 Mbytes
- Available hard drive space: 20 Mbytes
- For interfacing with C or C++: Microsoft Visual C++ 5.0 or later.

2.2 Installation

The development system comes in two flavors:

1. A console-based executable which is suitable to run from a DOS-prompt, from batch files, or under Emacs. See Section 2.4 [Command line editing], page 6.
2. A windowed executable providing command line editing and menus.

The distribution consists of a single, self-installing executable (`sp3w32.exe`) containing development system, runtime support files, library sources, and manuals.

Installed files on a shared drive can be reused for installation on other machines.

SICStus Prolog requires a license code to run. You should have received from SICS your site name, the expiration date and the code. This information is normally entered during installation:

```
Expiration date: ExpirationDate
Site: Site
License Code: Code
```

but it can also be entered later on by executing the following commands at a command prompt:

```
% splm -i Site
% splm -a sicstus3.8 ExpirationDate Code
```

2.3 Windows Notes

- Pre-linked foreign resources are not supported under Windows; they have to be implemented as DLLs. They are created using the utility `splfr` as described in the user's manual.
- The file name arguments to `splfr` and `spld` should not have embedded spaces. The reason for this is that not all C-compilers/linkers seem to support quoting of file name arguments.

2.3.1 Launching Runtime Systems on Target Machines

This section describes how to launch a runtime system on a so called *target machine*, i.e. a machine which does not have SICStus installed.

In order to locate all relevant files, the following directory structure is recommended.

```
myapp.exe
sprt38.dll
sp38\
+--- bin\
|    +--- sprt.sav
+--- library\
     +--- <files from %SP_PATH%\library>
```

`myapp.exe` is typically created by a call to `spld`:

```
% spld --main=user [...] -o ./myapp.exe
```

If the directory containing `sprt38.dll` contains a directory called `sp38`, SICStus assumes that it is part of a Runtime System as described in the picture. The runtime library (`sprt.sav`) is then looked up in the directory ('`sp38/bin`'), as in the picture. Furthermore, the initial `library_directory/1` fact will initially be set to the same directory with `sp38/library` appended.

The directory structure under `library/` should look like in a regular installed SICStus, including the platform-specific subdirectory (`x86-win32-nt-4` in this case). If your application needs to use `library(system)` and `library(random)`, your directory structure may look like:

```
myapp.exe
sprt38.dll
sp38\
+--- bin\
|    +--- sprt.sav
+--- library\
     +--- random.po
     +--- system.po
     +--- x86-win32-nt-4 \
          +--- random.dll
```

```
+--- system.dll
```

The `sp*` files can also be put somewhere else in order to be shared by several applications provided the `sprt38.dll` can be located by the DLL search.

The 38 in the file names above is derived from SICStus's major and minor version numbers, i.e. currently 3 and 8. Naming the files with version number enables applications using different sicstus versions to install the `sp*` files in the same directory.

2.3.2 Generic Runtime Systems

There are three ready-made runtime systems provided with the distributions, '`%SP_PATH%\bin\sprt.exe`', '`%SP_PATH%\library\sprtw.exe`', and '`%SP_PATH%\bin\sprti.exe`'. These are created using `spld`:

```
% spld --main=restore main.sav -o sprt.exe
% spld --main=restore main.sav -i -o sprti.exe
% spld --main=restore main.sav --window -o sprtw.exe
```

These are provided for users who do not have a C-compiler available. The programs launches a runtime system by restoring the saved state `main.sav` (created by `save_program/2`) and then call the predicate `runtime_entry/1` (defined by the user), setting the first argument to the atom `start`. Alternatively, the runtime system's entry point may be specified using `save_program/2`.

The program '`sprti.exe`' assumes that the standard streams are connected to a terminal, even if they to not seem to be (useful under Emacs, for example). '`sprtw.exe`' is a windowed executable, corresponding to '`spwin.exe`'.

For more info on how `spld` works, see section "The spld utility" in *SICStus Prolog Manual*.

2.3.3 Setting SP_PATH under Windows

The use of the `SP_PATH` variable under Windows is discouraged, since Windows applications can find out for themselves where they were started from.

`SP_PATH` is only used if the directory where `sprt<ver>.dll` is loaded from does not contain `sp<ver>` (a directory) or `sprt.sav` (where `<ver>` is "38" for SICStus version 3.8(.x)). If `SP_PATH` is used, SICStus expects it to be set such that `%SP_PATH%/bin` contains `sprt.sav`. See Section 2.3.1 [Launching Runtime Systems on Target Machines], page 5.

2.4 Command line editing

Command line editing supporting Emacs-like commands and IBMPC arrow keys is provided in the console-based executable. The following commands are available:

`^h` erase previous char

`^d`	erase next char
`^u`	kill line
`^f`	forward char
`^b`	backward char
`^a`	begin of line
`^e`	end of line
`^p`	previous line
`^n`	next line
`^i`	insert space
`^s`	forward search
`^r`	reverse search
`^v`	view history
`^q`	input next char blindly
`^k`	kill to end of line

Options may be specified in the file '`%HOME%\spcmd.ini`' as:

Option Value

on separate lines. Recognized options are:

`lines` *Value* is the number of lines in the history buffer. 1-100 is accepted; the default is 30.

`save` *Value* is either 0 (don't save or restore history buffer) or 1 (save history buffer in '`%HOME%\spcmd.hst`' on exit, restore history from the same file on start up.

The command line editing is switched off by giving the option '`-nocmd`' when starting SICStus. Command line editing will be automatically turned off if SICStus is run with piped input (e.g. from Emacs).

2.5 The console window

The console window used for the windowed executable is based on code written by Jan Wielemaker `<jan@swi.psy.uva.nl>`.

The stream-based console window is a completely separate library, using its own configuration info. It will look at the environment variable `CONSOLE` which should contain a string of the form *name*:*value*`{`,*name*:*value*`}` where *name* is one of:

`sl` The number of lines you can scroll back. There is no limit, but the more you specify the more memory will be used. Memory is allocated when data becomes available. The default is 200.

`rows`	The initial number of lines. The default is 24.
`cols`	The initial number of columns. The default is 80.
`x`	The X coordinate of the top-left corner. The default is `CW_USEDEFAULT`.
`y`	The Y coordinate of the top-left corner. The default is `CW_USEDEFAULT`.

You will normally specify this in your '`autoexec.bat`' file. Here is what an example:

```
% set CONSOLE=sl:600,x:400,y:400
```

2.6 Emacs Interface

Choosing ⟨EOF⟩ from the menu seems to generate an eternal ⟨EOF⟩ which is not useful for e.g. escaping a break level. Instead a *C-d* can be generated by typing *C-q C-d*.

2.7 Limitations

- File paths with both `/` and `\` as separator are accepted. SICStus returns paths using `/`. Note that `\`, since it is escape character, must be given as `\\` unless the prolog flag `character_escapes` is set to `off`.
- All file names and paths are converted to lowercase when expanded by `absolute_file_name/2` etc.
- File paths of the form `~/` are expanded using the values of the environment variable `HOME` or `HOMEDRIVE` and `HOMEPATH`. The form *~username/* is not expanded. The form *$VAR/* is expanded using the value of the environment variable *VAR*. The form *%VAR%/* is not recognized.
- Interruption by *^C* is limited on the windowed executable: *^C* is checked for upon character output and garbage collection only.
- Pre-linked foreign resources are not supported. The `--resources` option to `spld` is a no-op.
- In the windowed executable, the `user_error` stream is line buffered.
- Running under Emacs has been tried with GNU-Emacs v.19.31 and 19.34 (http://www.cs.washington.edu/homes/voelker/ntemacs.html). See above.
- Tcl/Tk: The `top_level_events` option to `tk_new/2` is not supported.
- `stream_select/3` is not supported.
- `stream_interrupt/3` is not supported.
- `library(timeout)` is not supported.
- `library(sockets)`: The `AF_UNIX` address family is (unsurprisingly) not supported; `socket_select/[5,6]` support only socket streams for arg 4(5).
- `library(system)`: `popen/3` is not supported. `kill/2` attempts to terminate the requested process irrespectively of the 2nd arg.

2.8 Files that may be redistributed with runtime systems

When a runtime system is redistributed to third parties, only the following files may be included in the distribution. All filenames are relative to '`%SP_PATH%`':

'`bin\sprt.sav`'

'`bin\jasper.jar`'

'`bin\*.dll`'

'`bin\*.po`'

'`library\*.{tcl,po}`'

'`library\*\*.dll`'

'`library\*\*.po`'

3 Tcl/Tk notes

Tcl/Tk itself is not included in the SICStus distribution. It must be installed in order to use the interface. It can be downloaded from the Tcl/Tk primary website:

http://www.scriptics.com

The Tcl/Tk interface module included in SICStus Prolog 3.8 (`library(tcltk)`) is verified to work with Tcl/Tk 8.2. *Previous* versions of the interface have been verified to work with Tcl/Tk versions 7.3/3.6, 7.4/4.0, 7.5/4.1, 7.6/4.2, 8.0, and 8.1. The current version of the interface may or may not work with these versions.

Under UNIX, the installation program automatically detects the Tcl/Tk version (if the user does not specify it explicitly).

Under Windows, the binary distribution is compiled against Tcl/Tk 8.2. If you need to use an older Tcl/Tk, contact SICStus Support.

The Tcl/Tk interface module is not supported under the following platforms: IRIX, Mac OS X Server.

3.1 The Tcl/Tk Terminal Window

The Tcl/Tk interface includes a experimental terminal window based on Tcl/Tk. It is opened by using the (undocumented) predicate:

`tk_terminal(Interp, TextWidget, InStream, OutStream, ErrStream)`
: Given a TextWidget, e.g. `.top.myterm`, this predicate opens three prolog streams for which the text widget acts as a terminal.

There is also a `library(tkconsol)`, making use of `tk_terminal/5`, which switches the Prolog top level to a Tk window. This is done by simply loading the library module.

4 Jasper notes

Jasper requires at least Java 2 (a.k.a. JDK 1.2) to run (the full development kit, not just the JRE). **Jasper does not work with Visual J++ or Visual Café**. Jasper is *only* supported under the following configurations:

Solaris 2.x (x86, SPARC)
: Verified using Sun's JDK 1.2.x, downloadable from
  http://java.sun.com/products/jdk/1.2/

Windows 95/98/NT
: Verified using Sun's JDK 1.2.2, downloadable from
  http://java.sun.com/products/jdk/1.2/

Linux (x86)
: Verified using Blackdown's JDK 1.2 (pre-release-v2, native threads, nojit). Downloadable from
  http://www.blackdown.org/java-linux.html

4.1 Getting Started

This section describes some tips and hints on how to get the interface started. This is actually where most problems occur.

Under Windows, it is recommended that you add SICStus's and Java's DLL directories to your `%PATH%`. This will enable Windows library search method to locate all relevant DLLs. For SICStus, this is the same as where `sicstus.exe` is located, usually `C:/Program Files/sicstus3/bin`). For Java it is usually `C:/jdk1.2.2/jre/bin/classic` and `C:/jdk1.2.2/bin`. For example:

```
set PATH=C:/Program Files/sicstus3/bin;C:/jdk1.2.2/jre/bin/classic;\
        C:/jdk1.2.2/bin;%PATH%
```

If SICStus is used as parent application, things are usually really simple. Just execute the query `| ?- use_module(library(jasper)).`. After that, it is possible to perform meta-calls (as described in the User's Manual), or load a foreign resource containing `foreign(...,java,...)` predicates.

On some platforms, you may encounter the following error message:

```
% sicstus
SICStus 3.8 (sparc-solaris-5.5.1): Wed Sep 22 08:42:14 MET DST 1999
Licensed to SICS
| ?- use_module(library(jasper)).
[...]
{SYSTEM ERROR: 'Attempted to load Java engine into sbrk\'d
SICStus system (try starting SICStus with -m option)'}
[...]
```

Since most platforms don't allow `sbrk()` and `malloc()` to coexist peacefully, SICStus refuses to load the JVM if not the `-m` flag was given to SICStus. The message can, as the error message suggests, be avoided if SICStus is started with the `-m` flag:

```
% sicstus -m
```

If Java is used as parent application, things are a little more complicated. There are a couple of things which need to be taken care of. The first is to specify the correct class path so that Java can find the Jasper classes (`SICStus`, `SPTerm`, and so on). This is done by specifying the pathname of the file `jasper.jar`:

```
% java -classpath $SP_PATH/bin/jasper.jar ...
```

`SP_PATH` does not need to be set; it is only used here as a placeholder. See the documentation of the Java implementation for more info on how to set classpaths.

The second is specify where Java should find the Jasper native library (`libjasper.so` or `jasper.dll`), which is loaded into the JVM by invoking the method `System.loadLibrary("jasper")`. This method uses a platform dependent search method to locate the Jasper native library, and quite often this method fails. A typical example of such a failure looks like:

```
% java -classpath [...]/jasper.jar se.sics.jasper.SICStus
Trying to load SICStus.
Exception in thread "main" java.lang.UnsatisfiedLinkError: no jasper
in java.library.path
at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1133)
at java.lang.Runtime.loadLibrary0(Runtime.java:470)
at java.lang.System.loadLibrary(System.java:745)
at se.sics.jasper.SICStus.loadNativeCode(SICStus.java:37)
at se.sics.jasper.SICStus.initSICStus(SICStus.java:80)
at se.sics.jasper.SICStus.<init>(SICStus.java:111)
at se.sics.jasper.SICStus.main(SICStus.java:25)
```

This can be fixed by explicitly setting the Java property `java.library.path` to the location of `libjasper.so` (or `jasper.dll`), like this:

```
% java -Djava.library.path=/usr/local/lib [...]
```

If this works properly, SICStus should have be loaded into the JVM address space. The only thing left is to tell SICStus where the runtime library (i.e. `sprt.sav`) is located. You may choose to specify this explicitly by either givin a second argument when initializing the `SICStus` object or by specifying the property `sicstus.path`:

Example (UNIX):

```
% java -Dsicstus.path=/usr/local/lib/sicstus-3.8
```

Example (Win32):

```
% java -Dsicstus.path="c:\Program Files\sicstus3"
```

If you do not specify any explicit path, SICStus will search for the runtime library itself.

If everything is setup correctly, you should be able to call `main` (which contains a short piece of test-code) in the SICStus root class, something like this:

```
% java -Djava.library.path=/usr/local/lib \
         -classpath /usr/local/lib/sicstus-3.8/bin/jasper.jar \
        se.sics.jasper.SICStus
Trying to load SICStus.
If you see this message, you have succesfully
initialized the SICStus Prolog engine.
```

It is similar under Win32, with the exception that the paths look slightly different.

4.2 Jasper and Native Threads

It is highly recommended that you run your JDK in native threads mode. This is the default under Windows. Under UNIX, most JDKs use native threads per default in version 1.2. The installation script `InstallSICStus` enables the user to disable native threads if desired (and if allowed by the JDK).

SICStus does not allow multiple native threads executing simultaneous in the emulator. In other words, all calls to SICStus have to be performed from the *same Java (native) thread*. If this condition is violated, a `IllegalCallerException` is thrown. There are also methods in the `SICStus` class to check whether the current thread is allowed to call SICStus methods, without throwing an exception. See the `se.sics.jasper` package documentation for more info.

The simplest way of avoiding `IllegalCallerException`s being thrown is to have a single, dedicated, Java-thread which is performs all the calls to SICStus. Other threads wanting to call Prolog can synchronize with the dedicated Java-thread using the methods `wait()` and `notify()`.

4.3 Known Bugs and Limitations in Jasper

- Jasper cannot be used from within applets, since Jasper relies on calling methods declared as `native`. This is due to a security-restriction enforced on applets; they are not allowed to call native code.
- If a green threads JVM is used (which is not recommended), avoid using terminal I/O routines in Java code called from Prolog (or in general when Prolog is the parent application). This is due to magic tricks the JVM does with file descriptors to handle blocking system calls without native threads.
- On some platforms you need to explicitly specify the `-native` option when calling `java`. The following error is an example of what may happen if you do not specify `-native`:

  ```
  % java -classpath .:[...]/lib/sicstus-3.8/bin/jasper.jar -Djava.library.path=[.
  ```

```
*** panic: libthread loaded into green threads
Abort (core dumped)
```

Instead, do

```
% java -native [...]
```

See your JDK documentation for more info on command-line parameters to the JVM.

- There is a known memory leak when using the argument conversion specifiers `[-chars]`, `[-string]`, `-chars`, and `-string`. They convert Java strings to UTF8 strings using `GetStringUTFChars()`, but they do not call `ReleaseStringUTFChars()` to release the string.

4.4 Java Examples Directory

There is an examples directory available in `$SP_PATH/library/jasper/examples`. See the file `README` for more info.

4.5 Resources

There are almost infinitely many Java resources on the Internet. Here is a list of a few which are related to Jasper and JNI.

- JavaSoft Homepage.
- JavaSoft's Java FAQ.
- JavaSoft Documentation Homepage.
- JNI Documentation.
- JavaSoft's JNI Tutorial.
- Yahoo's Java page.
- The ACM student magazine *Crossroads* has published an article on the JNI. This article may be out of date.

5 Visual Basic notes

The Visual Basic - SICStus Prolog interface consists of the following files:

- `vbsp.dll` (installed as '`SICStus\bin\vbsp.dll`')
- `vbsp.po` (installed as '`SICStus\bin\vbsp.po`')
- `vbsp.bas` (installed as '`SICStus\library\vbsp.bas`')

In order to use the interface, perform the following steps:

- include the file '`vbsp.bas`' in your Visual Basic project.
- put the files '`vbsp.dll`' and '`vbsp.po`' in a place where DLLs are searched for (For example the same directory as your applications EXE file or the Windows-System directory). This is true by default if '`SICStus\bin\`' is in the PATH environment variable.
- make the SICStus runtime DLL etc. available. See section "Launching Runtime Systems on Target Machines" in *SICStus Prolog Release Notes*.

6 Berkeley DB notes

As of SICStus 3.8, the library module `library(db)` has been replaced by `library(bdb)`. The functionality is similar, but `library(bdb)` is built on top of Berkeley DB. Berkeley DB can be downloaded from:

http://www.sleepycat.com

`library(bdb)` has been verified to work using Berkeley DB version 2.7.5.

When using Berkeley DB on Windows, you may want to set `%PATH%` to contain the path to `libdb.dll`. Consult the Berkeley DB documentation for further info.

7 The Emacs Interface

The Emacs Interface was originally developed for GNU Emacs 19.34 and is presently being maintained using XEmacs 21.1 and tested with GNU Emacs 19.34.1. For best performance and compatibility and to enable all features we recommend that the latest versions of GNU Emacs or XEmacs are used.

7.1 Installation

Starting with SICStus 3.8 the Emacs interface is distributed with SICStus and installed by default. The default installation location for the emacs files is '`<prefix>/lib/sicstus-3.8/emacs/`' on UNIX platforms and '`C:/Program Files/SICStus/emacs/`' on Windows.

For maximum performance it is recommended, that the Emacs lisp files (extension .el) are compiled. This can be done from within Emacs with the command *`M-x byte-compile-file`*. See section "Installation" in *SICStus Prolog Manual*, for further details.

7.1.1 Installing On-Line Documentation

It is possible to look up the documentation for any built in or library predicate from within Emacs (using *`C-c ?`* or the menu). For this to work Emacs must be told about the location of the '`info`'-files that make up the documentation. This can be done for the entire emacs installation or on a per user basis, see section "Installation" in *SICStus Prolog Manual*, for further details.

The default location for the '`info`'-files are '`<prefix>/lib/sicstus-3.8/doc/info/`' on UNIX platforms and '`C:/Program Files/SICStus/doc/info/`' on Windows.

More recent versions of GNU Emacs and XEmacs should be able to automatically incorporate info files from a subdirectory into the main Info documentation tree. It is therefore recommended that the SICStus Info files are kept together in their own directory.

8 Revision history

This chapter summarizes the changes in release 3 wrt. previous SICStus Prolog releases as well as changes introduced by patch releases.

8.1 Changes in release 3

- Backslashes (\) in strings, quoted atoms, and integers written in '`0`'' notation denote escape sequences. Character escaping can be switched off.
- Multifile declarations are required in *all* files where clauses to a multifile predicate are defined. This complies with the ISO Prolog Standard.
- The built-in predicate `call_residue/2` has been modified so that goals that are disjunctively blocked on several variables are returned correctly in the second argument.
- The built-in predicate `setarg/3` has been removed. Its functionality is provided by the new built-ins `create_mutable/2`, `get_mutable/2`, `update_mutable/2`, and `is_mutable/2`, which implement a timestamp technique for value-trailing with low-level support.
- The built-in predicates `unix/1` and `plsys/1` have been removed. Their functionality is provided by `prolog_flag(argv,`*X*`)`, by the new `halt/1` built-in, and by the new `library(system)` module which also contains several new predicates.
- The socket I/O built-ins have been moved to the new `library(sockets)` module.
- The built-in `time_out/3` has been moved to the new `library(timeout)` module.
- The built-ins `term_hash/[2,4]`, `subsumes_chk/2`, and `term_subsumer/3` have been moved to the new `library(terms)` module, which also contains operations for unification with occurs-check, testing acyclicity, and getting the variables of a term.
- The foreign language interface (Prolog-to-C) has been extended with the types `+chars`, `-chars` and `[-chars]` for fast conversion between C strings and Prolog lists of character codes. Several new interface functions are available.
- The memory handling of the C-to-Prolog interface has been simplified by passing each Prolog term as a "handle" object, called an `SP_term_ref`, making the functions `SP_show_term()` and `SP_hide_term()` obsolete.
- The InterViews 2.6 based GUI module `library(gmlib)` has been replaced by the Tcl/Tk
  based `library(tcltk)`. A version of `library(gmlib)` converted to SICStus Prolog release 3 is available from '`ftp://ftps.ics.se/archive/sicstus3/gmlib.tar.gz`'.
- The `library(objects)` module has been enhanced.
  * Inheritance is static, i.e. determined at object creation time, and is implemented as module importation.
  * A new, very light-weight, type of object: *instance*.
  * *Attributes*, efficient storage of terms in objects.
  * Unprefixed goals in methods denote message passing to `self`. Prolog goals in methods must be prefixed by `:`.

- In `library(charsio)`, the `open_chars_stream/[3,4]` predicates have been replaced by `open_chars_stream/2` and `with_output_to_chars/[2,3]`.
- The `library(assoc)` module now implements AVL trees instead of unbalanced binary trees.
- The new `library(atts)` implements attributed variables, a general mechanism for associating logical variables with arbitrary attributes. Comes with a number of hooks that make it convenient to define and interface to constraint solvers.
- The Boolean constraint solver has been moved to the new `library(clpb)` and is implemented on top of `library(atts)`.
- New constraint solvers for rationals (`library(clpq)`) and reals (`library(clpr)`), implemented on top of `library(atts)`.
- `user:goal_expansion/3` is a new hook predicate for macro-expansion.
- `bb_put/2`, `bb_get/2`, `bb_delete/2`, and `bb_update/3` are new built-ins implementing blackboard primitives.
- `prolog_load_context/2` is a new built-in predicate for accessing aspects of the context of files being loaded.
- `user:file_search_path/2` is a new hook predicate providing an alias expansion mechanism for filenames.
- `gcd/2` is a new built-in function.
- The statistics keyword `walltime` measures elapsed absolute time.
- In runtime systems, `ensure_loaded/1` and `use_module/[1,2,3]` have the same semantics as in development systems.
- Native code compilation available for MIPS platforms.
- Problems in native code compilation for certain SPARC models have been eliminated.
- Performance improvements include emulated code speed, native code speed, and the foreign language interface.
- The system has been ported to the DEC OSF/1 Alpha (a 64-bit platform).

8.2 Changes introduced in 3#4

- New built-in predicates and shell commands for creating and loading foreign language modules and creating customized development and runtime systems. Previous built-ins remain for backwards compatibility.
- Slight changes in the C interface: hook variables are set by function calls, `SP_foreign_reinit_hook` is not supported.
- The system has been ported to the Microsoft Win32 platform (Intel x86).
- The system has been ported to the Macintosh.
- The system has been ported to the OS/2 (32bit) platform (Intel x86).
- If the init file '`~/.sicstusrc`' is not found, SICStus looks for '`~/sicstus.ini`'.
- `library(sockets)`: `socket_select/5` arg 1 may be a, possibly empty, list of passive sockets, arg 3 returns a, possibly empty, list of new streams.

- `library(system)`: The following new predicates are provided: `tmpnam/1`, `directory_files/2`, `file_property/2`, `delete_file/2`, `make_directory/1`.
- A new constraint solver for finite domains (`library(clpfd)`), implemented on top of `library(atts)`.

8.3 Changes introduced in 3#5

- New built-in `open/4`, enables opening files in binary mode.
- `library(charsio)`: New predicate `with_output_to_chars/4`.
- `library(heaps)`: New predicates `delete_from_heap/4`, `empty_heap/1`, `is_heap/1`.
- `library(queues)`: New predicate `is_queue/1`.
- `library(sockets)`: New predicates: `socket_accept/3`, and `socket_select/6` provide address of connecting client. `hostname_address/2` resolves name/ip-number.
- `SP_atom_length` returns the print name length of a Prolog atom.
- Modification time instead of current time stored for loaded files.

8.4 Changes introduced in 3#6

- `toplevel_print_options` and `debugger_print_options` are new Prolog flags controlling the toplevel's and debugger's printing behavior.
- `is_mutable/1` is a new built-in which is true for mutables.
- '`~@`' is a new spec in `format/[2,3]` for arbitrary goals.
- Mutables are initialized correctly when read in.
- `toplevel_print_options` and `debugger_print_options` are new Prolog flags controlling the toplevel's and debugger's printing behavior.
- `is_mutable/1` is a new built-in which is true for mutables.
- '`~@`' is a new spec in `format/[2,3]` for arbitrary goals.
- Mutables are initialized correctly when read in.
- The finite domain constraint solver (`library(clpfd)`) has been enhanced by a programming interface for global constraints, improved compilation to library constraints and other performance enhancements, and by a number of new exported constraints.
- `library(objects)`: New hook predicate `user:method_expansion/3`.
- `library(sockets)`: `socket_select/5` has extended functionality.
- Efficiency bugs in `format/[2,3]` fixed.
- Bug in `save_program/[1,2]` with native code fixed.
- Bugs in `library(chr)` fixed, and a couple of new constraint handlers fixed.
- A problem with source linked debugging of DCG rules fixed.
- Prevent looping on duplicates in `module/2` decl.
- Prevent memory overrun in `library(tcltk)`.

8.5 Changes introduced in version 3.7

- The concept of patchlevels removed and replaced by versions.
- `library(chr)`: A new library module providing Constraint Handling Rules; see
  `http://www.pst.informatik.uni-muenchen.de/~fruehwir/chr-solver.html`
- *Jasper*, a bi-directional Java-interface, consisting of extensions to the existing FLI and a new library module `library(jasper)`.
- Atom garbage collection, invoked by `garbage_collect_atoms/0`, and controlled by the `agc_margin` Prolog flag. New statistics options: `atoms`, `atom_garbage_collection`. New interface functions: `SP_register_atom`, `SP_unregister_atom`.
- Calls with clean-up guaranteed, provided by `call_cleanup/2`, which replaces `undo/1`.
- Source-linked debugging, controlled by the `source_info` Prolog flag.
- Debugger enhancements: tracing of compiled code; a new debugger mode `zip` and built-ins `zip/0`, `nozip/0`; new debugger commands *out n*, *skip i*, *quasi-skip i*, *zip*, *backtrace n*, *raise exception*. Modules can be declared as *hidden* which disables tracing of their predicates.
- Saved states are available in runtime systems, and are portable across platforms and between development and runtime systems. `save/[1,2]` are gone. In most cases, `save_program/2` can be used in their place, with a little rearrangement of your code. Predicates can be declared as *volatile*.
- A interface function `SP_restore` is the C equivalent of `restore/1`, which now only restores the program state, leaving the Prolog execution stacks unchanged.
- The GNU Emacs interface was enhanced: source-linked debugging, new menus, speed, help functions, electric functions, indentation, portability, bug fixes.
- The reader can return layout information about terms read in. New `read_term/3` option: `layout(-`*Layout*`)`. New hook predicate: `user:term_expansion/4`.
- Module name expansion of goals is done prior to execution of meta-calls.
- Imported predicates can be spied and abolished.
- `random:randset/3` returns a set in standard order.
- `db:db_canonical/[2,3]` are new; can be used to check whether two *TermRef*s refer to the same term.
- `clpfd:serialized_precedence/3` and `clpfd:serialized_precedence_resource/4` are new; model non-overlapping tasks with precedence constraints or sequence-dependent setup times.
- In object method bodies, goals of the form `:`*Goal* are translated according to the manual. Earlier versions treated arguments occurring in the '`:`' position of meta-predicates specially.
- A new interface function `SP_raise_fault` and interface macro `SP_on_fault` are available for handling runtime faults that cannot be caught as exceptions.
- A new interface function `SP_set_memalloc_hooks` is available for redefining the memory manager's bottom layer. Related to that, there is a new command-line option '`-m`'.

- Development and runtime systems have been reorganized internally. All use a runtime kernel shared object or DLL, and are initialized by restoring saved states. Development systems additionally use a development kernel shared object or DLL.
- The '`-B`' command-line option is gone in the start-up script, and some new options have appeared.
- Under UNIX: new option '`-base`' to override the executable used by the start-script.
- Under UNIX: improvements in the configure-script; better options to specify Tcl/Tk versions and paths.
- Hookable standard-streams.
- Floating-point operations on Digital Alpha are now IEEE-conformant.
- `reinitialise/0` does not load any initialization files given in '`-i`' or '`-l`' command line flags.
- Under UNIX: New option `-S` to `spmkrs` and `spmkds` to link the SICStus Runtime Kernel (and development extensions for `spmkds`) statically into the executable.
- ?- `[`*File1*`,`*File2*`,...]` was broken.
- `require/1` did not find all directories.
- Runtime systems could crash after GC.
- Bugs in `clp[qr]:dump/3`, `clp[qr]:expand/0`, `clp[qr]:noexpand/0`.
- The garbage collector reported too many bytes collected.
- Memory overflows were not handled gracefully.
- Imported predicates couldn't be abolished.
- `arrays:arefa/3`, `arrays:arefl/3`, `heaps:min_of_heap/5` are now steadfast.
- Most `library(clpfd)` predicates now check the type of their arguments. Bugs fixed in `relation/3`, `serialized/2`, `all_distinct/1`.
- `frozen/2` could crash on an argument of the wrong type.
- `SP_get_list_n_chars` does not require a proper list.
- Problems with exceptions in embedded commands in source files.
- Problems with `load_files(`*Files*`, [compilation_mode(assert_all)])`.
- For `load_files(`*Files*`, [if(changed)])`, a non-module file is not considered to have been previously loaded if it was loaded into a different module.
- Incorrect translation of `if/3` goals in DCG rules.
- On Win32, `system:mktemp/2` sometimes returned filenames with backslashes in them.

8.6 Changes introduced in version 3.7.1

- The type-specifier `object` in Jasper has changed to `object(Class)`.
- Under UNIX: Error-handling in `splfr`, `spmkrs`, `spmkds`.
- Jasper did not convert return values correctly when calling Java from Prolog.
- Jasper did not handle instance methods correctly.
- Some of the legal type-specifiers in Jasper were rejected by the glue-code generator.

- Efficiency bugs in `format/[2,3]` fixed.
- Bug in `save_program/[1,2]` with native code fixed.
- Bugs in `library(chr)` fixed, and a couple of new constraint handlers fixed.
- A problem with source linked debugging of DCG rules fixed.
- Prevent looping on duplicates in `module/2` decl.
- Prevent memory overrun in `library(tcltk)`.

8.7 Changes introduced in version 3.8

8.7.1 Wide character support

Wide character handling is introduced, with the following highlights:

- character code sets up to 31 bit wide;
- three built-in wide character modes (ISO_8859_1, UTF8, EUC), selectable via environment flags;
- complete control over the external encoding via hook functions.

For programs using the default ISO_8859_1 character set, the introduction of wide characters is transparent, except for the string format change in the foreign interface, see below.

In programs using the EUC character set, the multibyte EUC characters are now input as a single, up to 23 bit wide, character code. This character code can be easily decomposed into its constituent bytes, if needed. The encoding function is described in detail in the SICStus manual.

To support wide characters, the foreign interfaces now use UTF-8 encoding for strings containing non-ASCII characters (codes >= 128). This affects programs with strings that contain e.g. accented characters and which transfer such strings between Prolog and C. If such a string is created on the C side, it should be converted to UTF-8, before passing it to Prolog. Similarly for a string passed from Prolog to C, if it is to be decomposed into characters on the C side, the inverse transformation has to be applied.

Utility functions `SP_code_wci` and `SP_wci_code` are provided to support the conversion of strings between the WCI (Wide Character Internal encoding, i.e. UTF-8) format and wide character codes.

8.7.2 Breakpointing debugger

A new general debugger is introduced, with advanced debugging features and an advice facility. It generalizes the notion of *spypoint* to that of the *breakpoint*. Breakpoints make it possible to e.g. stop the program at a specified line, or in a specified line range, or to call arbitrary Prolog goals at specified ports, etc. Highlights:

- Advice facility — useful for non-interactive debugging, such as checking of program invariants, collecting information, profiling, etc.

- Debugger hook predicate — new interactive tracer commands can be defined.
- Tracer information access — data on current and past execution states, such as those contained in the ancestor list, or the backtrace, is now accessible to the program.
- The following built-in predicates have been added: `add_breakpoint/2`, `spy/2`, `current_breakpoint/4`, `remove_breakpoints/1`, `disable_breakpoints/1`, `enable_breakpoints/1`, `execution_state/1`, and `execution_state/2`. `user:debugger_command_hook/2` is a new hook predicate.

  The predicates `nospy/1` and `nospyall/0` have slighty changed meaning.

  The predicate `spypoint_condition/3` has been removed.

8.7.3 ISO compliance

SICStus 3.8 supports standard Prolog, adhering to the International Standard ISO/IEC 13211-1 (PROLOG: Part 1—General Core). At the same time it also supports programs written in earlier versions of SICStus. This is achieved by introducing two execution modes `iso` and `sicstus`. Users can change between the modes using the prolog flag `language`. Main issues:

- The `sicstus` execution mode is practically identical to 3.7.1, except for minor changes in error term format.
- The `iso` mode is fully compliant with ISO standard, but no strict conformance mode is provided.
- The dual mode system supports the gradual transition from legacy SICStus code to ISO Prolog compliant programs.
- Note that the built-in predicates, functions and Prolog flags, required by the ISO standard, are also available in `sicstus` execution mode, unless they conflict with existing SICStus predicates or functions. This expansion of the language carries a remote risk of name clashes with user code.

8.7.4 Generic new features

- The `spmkds` and `spmkrs` utilities for creating stand-alone executables have been replaced by a common `spld` utility which takes several new options. Runtime systems do not always need a main program in C. On Windows, the resulting executable can optionally be windowed. The `splfr` utility takes several new options. The development and runtime kernels have been merged into a single one.
- Partial saved states corresponding to a set of source files, modules, and predicates can be created by the new built-in predicates `save_files/2`, `save_modules/2`, and `save_predicates/2` respectively. These predicates create files in a binary format, by default with the prefix '`.po`' (for Prolog object file), which can be loaded by `load_files/[1,2]`. The `load_type(Type)` option of `load_files/2` has been extended. Partial saved states render '`.ql`' files obsolescent.
- The new built-in predicate `trimcore/0` reclaims any dead clauses and predicates, defragmentizes Prolog's memory, and attempts to return unused memory to the operating system. It is called automatically at every top level query.

- The value of the new read-only Prolog flag `host_type` is an atom identifying the platform, such as `'x86-linux-glibc2.1'`.
- The functionality of the `source_info` Prolog flag, introduced in release 3.7, has been extended beyond the Emacs interface. Line number information is now included in error exceptions whenever possible. This information is displayed in debugging and error messages (outside Emacs) or causes Emacs to highlight the culprit line of code. Valid values are `off`, `on`, and `emacs`.
- Predicate indicators can take the form *Name/[Arity,...,Arity]* in `spy/[1,2]`, `nospy/1`, `listing/1`, `abolish/1`, `profile_data/4`, `profile_reset/1`, `save_predicates/2`, and `gauge:view/1`.
- The new interface functions `SP_chdir()` and `SP_getcwd()` provide access to the current working directory.
- The interface function `SP_load()` has been generalized to correspond to `load_files/1`.
- The interface function `SP_deinitialize()` is now documented.
- Windows: the registry is no longer used by SICStus itself. The SICStus Runtime Library is located based on the location of `sprt<xx>.dll`. `SP_PATH` is only used as a last resort. See Section 2.3 [Windows notes], page 5.
- Source code compilation and installation procedure has been improved and simplified. See '`README`' and '`INSTALL`' in the source distribution for documentation.
- The layout of the Gauge graphical user interface has been improved.
- The new `library(bdb)` provides an interface to the Berkeley DB toolset for persistent storage, and replaces `library(db)`. The programming interface of the new module is similar to that of the old one, with some new concepts added such as *iterators*. The sources of the old library module are available from:

  ```
  ftp://ftp.sics.se/archive/sicstus3/libdb.tgz
  ```

- `library(db)` is obsolete and will be removed in the next major release.
- Generic runtime systems on Windows are built using `spld` and exist in three flavors: generic character based (`sprt.exe`), generic character based interactive (`sprti.exe`), and generic windowed (`sprtw.exe`). See Section 2.3.2 [Generic Runtime Systems], page 6.
- The manual chapter for `library(tcltk)` has been rewritten and greatly expanded.
- `library(clpq)` and `library(clpr)`: new predicates `inf/4` and `sup/4`.
- Code fragments loaded via the Emacs interface are imported into the type-in module, unless the source file has an explicit mode line.
- `library(gcla)` has been removed.
- `initialization/[0,1]` have been replaced by ISO compliant initializations.

8.7.5 New features in library(jasper)

- Java 2 (a.k.a. JDK 1.2) is now required. `library(jasper)` will not work using JDK 1.1.x.
- Support for native threads JDKs. See Section 4.2 [Jasper and Native Threads], page 13.

- Changed package name from `jasper` to `se.sics.jasper`, according to JavaSoft guidelines. See Section 4.1 [Getting Started], page 11.
- Classfiles are now placed in `jasper.jar`, which is located in `$SP_PATH/bin`. See Section 4.1 [Getting Started], page 11.
- The shared library for Jasper (`jasper.dll` or `libjasper.so`) is now located in the same directory as the runtime kernel (default `<installdir>/lib` under UNIX, `<installdir>/bin` under Windows). See Section 4.1 [Getting Started], page 11.
- Meta-call functionality added (`jasper_call_instance/6`, `jasper_call_static/6`, etc.). This makes it possible to call Java without having to generate any glue-code (i.e. without a C-compiler).
- Support for handling local global references from Prolog (`jasper_create_global_ref/3`, `jasper_delete_global_ref/2`, `jasper_delete_local_ref/2`).
- `SPException.term` declared `protected` instead of `private`.
- New class `SPCanonicalAtom` to handle canonical representations of atoms and to make sure that they are safe with atom-gc. New methods `getCanonicalAtom` and `putCanonicalAtom`. New constructor for `SPPredicate`. `getAtom` and `putAtom` deprecated.
- New exception: `IllegalCallerException` is thrown if the current thread is not allowed to call SICStus.

8.7.6 New features in library(clpfd)

- `fd_degree/2` is new; returns the number of constraints attached to a variable.
- `labeling/2` requires the list of domain variables to have bounded domains. User-defined variable and value choice heuristics can be provided.
- `element/3` is interval-consistent in its second and third arguments. Use `relation/3` if domain-consistency is required.
- `serialized/3` is new and replaces `serialized_precedence/3` and `serialized_precedence_resource/4`. A number of new options control the algorithm. The space complexity no longer depends on the domain size.
- `cumulative/5` is new and takes the same options as `serialized/3`.
- `all_different/2`, `all_distinct/2` and `assignment/3` are new and take options controlling the algorithms.
- Generally, performance and error checking have been improved.

8.7.7 Bugs fixed in version 3.8

- `absolute_file_name/2`: could crash under IRIX; nested compound terms allowed
- `call_cleanup/2`: efficiency
- `close/1`: efficiency; handling the standard streams
- `format/[2,3]`: `~N` didn't work as expected; are now meta-predicates—needed by the `~@` format spec
- `load_files/[1,2]`: avoid changing directory; don't loop on duplicate exports

- `load_foreign_resource/1`: filenames containing periods on WinNT
- `print_message/2`: in runtime systems
- `prolog_load_context/2`: value of `term_position`
- `reinitialise/0`: sequencing of events
- `save_program/[1,2]`: fastcode handling; file mode creation masks; in runtime systems
- `write_term/[1,2]`: the `indented(true)` option and non-ground terms
- `library(db)`: efficiency of term deletion
- `library(heaps)`: `delete_from_heap/4`
- `library(objects)`: the `new/2` method; cyclic dependencies
- `library(random)`: determinacy and efficiency
- `library(sockets)`: noisy startup on Windows; block buffering is now the default
- `library(system)`: `sleep/1` admits floats as well as integers
- `library(terms)`: `subsumes_chk/2` and `variant/2` now don't unblock goals
- glue code generator: incorrect translation of `+chars`; syntax error messages were suppressed
- all system messages go via the `print_message/2` interface
- input argument checking is generally stricter
- resources are unloaded in LIFO order but loaded in FIFO order at save/restore
- CLP(Q,R): answer constraint projection
- problems with bignums and big terms in '`.ql`' files
- detecting invalid goals in metacalls, asserts, `load_files/[1,2]`
- spurious redefinition warnings
- bignum quotient/remainder on 64-bit architectures
- compiler: complexity of compiling multiple clauses with same key, code generation quality for inline goals
- memory manager: avoiding dangling pointers on Windows, better reclamation of dead clauses and predicates, using dynamic hashing and *hashpjw* for atoms, keeping predicate tables as small as possible, avoiding stack overflow if multiple goals get simultaneously unblocked, better reuse of free memory blocks
- garbage collector: removing redundant trail entries for mutables, improved scope and speed of generational garbage collection
- callbacks to Prolog while reading from the terminal
- printing atoms with character codes in 27...31
- reading atoms with `\c`
- Floating point NaN (Not a Number). Now behaves consistently across platforms. In particular fixed Windows related bugs with arithmetic on and printing of NaN.
  - Arithmetic comparisons involving NaN now fails (except `=\=`). Note that `X is nan, X =:= X` fails.
  - Term order for NaN is now defined and the same for all platforms. There is a single NaN and it lies between (the float) +inf and the integers.

9 Generic limitations

On 32-bit architectures, the total data space cannot exceed 256 Mb. The Linux implementation of `sbrk()` returns memory starting at `0x08000000`, so in practice the limit there is 128 Mb.

The number of arguments of a compound term may not exceed 255.

The number of atoms created may not exceed 262143.

The number of characters of an atom may not exceed 65535.

NUL is not a legal character in atoms.

There are 256 “temporary” and 256 “permanent” variables available for compiled clauses.

Saved states are not portable between 32-bit and 64-bit architectures, or from a system built with native code support to a system without native code support for the same architecture.

Indexing on big integers or floats is coarse.

10 Questions and answers

Current support status for the various platforms can be found at the SICStus Homepage:

http://www.sics.se/sicstus/

Information about and fixes for bugs which have shown up since the latest release can be found there as well.

Send requests for ordering information to

sicstus-request@sics.se

Send bug reports to

sicstus-support@sics.se

Bugs tend actually to be fixed if they can be isolated, so it is in your interest to report them in such a way that they can be easily reproduced.

The mailing list

sicstus-users@sics.se

is a moderated mailing list for communication among users and implementors. To [un]subscribe, write to

sicstus-users-request@sics.se

Table of Contents