SICStus Prolog Release Notes

by the Intelligent Systems Laboratory

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

Release 3.9
February 2002

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

Copyright © 2002 SICS

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

Permission is granted to make and distribute verbatim copies of these notes provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of these notes under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of these notes into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by SICS.

1 Supported platforms

Release 3.9 is actively being supported on the following platforms:

- Compaq Tru64 UNIX V5 (formerly Digital Unix) on ALPHA.
- Linux Intel
- HP-UX 11.x
- IRIX 6.5
- Solaris 7 SPARC 32/64 bit
- Solaris 7 Intel
- Windows 95/98/ME/NT/2000/XP
- FreeBSD 4.4
- AIX 4.3
- MacOS X 10.x

In addition, earlier versions of SICStus have been verified to run on a number of other platforms. Contact sicstus-support@sics.se if you have any questions about a particular platform (listed or not listed).

2 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 2.2 [UNIX installation], page 2. For example:

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

2.1 The Crypt Tool

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 tool available via anonymous FTP from

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

The enclosed README file describes how to compile it.

2.2 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-support@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.

2.3 Foreign Language Interface

2.3.1 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 `spconfig-`*version*. Follow these instructions:

1. Locate the configuration file `spconfig-`*version*. It should be located in the same directory as `splfr` and `spld`.
2. Make a copy for `spconfig-`*version*; let's 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 `spconfig-`*version* 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.

2.3.2 How to Create Dynamic Linked Foreign Resources Manually

The only supported method for building foreign resources are by compiling and linking them with `splfr`. However, this is sometimes inconvenient, for instance when writing a Makefile for use with UNIX `make`. To figure out what needs to be done to build a foreign resource, you should build it once with `splfr --verbose --keep ...`, note what compiler and linker flags are used, and save away any generated files. You can then mimic the build commands used by `splfr` in your Makefile, or similar. You should repeat this process each time you upgrade SICStus, even if only the revision part of the SICStus version changes.

2.3.3 Interfacing to C++

Functions in C++ files which should be called from Prolog must use C linkage, 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.

2.3.4 Runtime Systems on Target Machines

This section describes how to distribute runtime systems on *target machines*, i.e. machines which do not have SICStus installed. An alternative approach to what is presented here is to package the whole application as an all-in-one executable, section "All-in-one Executables" in *the SICStus Prolog Manual*.

In order to build a runtime system for distribution on a target machine, the option `--moveable` must be passed to `spld`. This option prevents `spld` from hardcoding any (absolute) paths into the executable.

Next, in order for SICStus to be able to locate all relevant files, the following directory structure should be used.

```
myapp.exe
lib/
+--- libsprt39.so
+--- sicstus-3.9.0/
     +--- bin/
     |    +--- sprt.sav
     +--- library/
          +--- <files from $SP_PATH/library>
```

If support for multiple SICStus instances is needed, then the run-times named e.g. '`libsprt39_instance_01_.so`', need to be available as well, in the same place as '`libsprt39.so`'.

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

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

On most platforms, the above directory layout will enable the executable to find the SICStus run-time ('`libsprt39.so`') as well as the boot file '`sprt.sav`'. In addition, application specific files, e.g. a `.sav` file, can be found using the automatically set environment variables `SP_APP_DIR` or `SP_RT_DIR`. Currently, this is the case on Solaris and Linux and, using a slightly different directory layout, on Win32 (see Section 3.3.1 [Runtime Systems on Target Machines], page 7).

On some platforms, the executable will not be able to locate '`sprt.sav`' unless the environment variable `SP_PATH` is set. If the example above is rooted in '`/home/joe`', then `SP_PATH` should be set to '`/home/joe/lib/sicstus-3.9.0`'. This is not needed on platforms where `SP_APP_DIR` or `SP_RT_DIR` can be used, e.g. on Linux, Solaris or Win32.

Unless the `--static` option is passed to `spld`, it might also be necessary to set `LD_LIBRARY_PATH` (or equivalent) to '`/home/joe/lib`' (in the example above) in order for the dynamic linker to find '`libsprt39.so`'. If the `--static` option is used, this is not necessary. Setting `LD_LIBRARY_PATH` is not needed, nor recommended, on Linux and Solaris (and Win32).

2.4 Platform Specific Notes

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

- **Solaris 64bit:** You cannot install (or build) the 64 bit version of SICStus using gcc. You need to use the Sun Workshop/Forte compiler. InstallSICStus will try to find it during the install but if that fails, you can set the environment variable `CC` to e.g. '`/opt/SUNWspro/bin/cc`' before invoking InstallSICStus.
- **Solaris 64bit:** The following libraries are not supported: `library(bdb)`, `library(tcltk)`, `library(jasper)`.
- **Solaris SPARC** The memory management bottom layer is based on `malloc()/MM_USE_MALLOC` instead of the usual `sbrk()/MM_USE_SBRK`. The latter did not work reliably. This is as if `sicstus -m` was always used.

2.5 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-3.9.0`':

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

'`bin/sprt.sav`'

'`bin/jasper.jar`'

'`library/*.{tcl,po,pl}`'
: Except '`license.pl`'!

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

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

'`sp_platform`'
: (Located with `InstallSICStus`)

3 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 (typically `C:\Program Files\SICStus Prolog 3.9.0`. Here, `%SP_PATH%` is just a place-holder; you usually do not need to set the environment variable `SP_PATH`, but see Section 3.3.3 [Setting SP_PATH under Windows], page 9. For example:

```
C:\> set PATH=C:\Program Files\SICStus Prolog 3.9.0\bin;%PATH%
```

You may also want to include the paths to Tcl/Tk (see Chapter 4 [Tcl/Tk Notes], page 13), Java (see Section 5.2 [Getting Started], page 15), and Berkeley DB (see Chapter 7 [Berkeley DB notes], page 23).

3.1 Requirements

- Operating environment: Microsoft Windows 95, 98, ME, NT 4.0, 2000, XP. Windows 2000 or newer is recommended.
- Processor: 386 or better.
- Available user memory: 16 Mbytes, more is recommended.
- Available hard drive space: 20 Mbytes (approximate)
- For interfacing with C or C++, or for using `spld` or `splfr`: Microsoft Visual C++ 6.0 or later.

3.2 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 3.4 [Command line editing], page 9.
2. A windowed executable providing command line editing and menus.

The distribution consists of a single, self-installing executable (‘`InstallSICStus.exe`’) containing development system, runtime support files, library sources, and manuals. Note that the installer itself asks for a password, when started. This is different from the license code.

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 sicstus3.9 ExpirationDate Code
```

on Windows NT/2000/XP `splm` must be run by a user with Administrative rights.

3.3 Windows Notes

- The file name arguments to `splfr` and `spld` should not have embedded spaces. For file names with spaces, you can use the corresponding short file name.
- Windows 95/98/ME: The shortcut installed in the '`Start`' menu (e.g. '`Start\Programs\SICStus Prolog 3.9.0`') may not work immediately after installation. Restarting after installing SICStus appears to cure this. If this does not help, you can add your own shortcut to e.g. '`C:\Program Files\SICStus Prolog 3.9.0\bin\spwin.exe`'.
- Selecting the '`Manual`' or '`Release Notes`' item in the '`Help`' menu may give an error message similar to '`... \!Help\100#!Manual.lnk could not be found`'. This happens when Adobe Acrobat Reader is not installed or if it has not been installed for the current user. Open '`C:\Program Files\SICStus Prolog\doc\pdf\`' in the explorer and try opening '`relnotes.pdf`'. If this brings up a configuration dialog for Adobe Acrobat, configure Acrobat and try the '`Help`' menu again. Alternatively, you may have to obtain Adobe Acrobat. It is available for free from http://www.adobe.com/.
- Windows NT, 2000 and later: We recommend that SICStus is installed by a user with administrative privileges and that the installation is made '`For All Users`'.

  If SICStus is installed for a single user, then SICStus will not find the license information when started by another user. In this case, you can use the command line tool '`splm.exe`' as described in the message containing your license code. Currently '`splm.exe`' must be run as a user with Administrative rights.
- The first time the installer is run, it will install necessary system files for supporting the new '`Windows Installer`' technology from Microsoft. This will fail unless the user has administrative rights. A typical symptom is an error message asking for '`msiexec`'. The Windows Installer technology is already part of Windows 2000 and later.

3.3.1 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. An alternative approach to what is presented here is to package the whole application as an all-in-one executable, section "All-in-one Executables" in *the SICStus Prolog Manual*.

In order to locate all relevant files, the following directory structure should be used.

```
myapp.exe
sprt39.dll
```

```
sp39\
+--- bin\
|    +--- sprt.sav
+--- library\
     +--- <files from %SP_PATH%\library>
```

if support for multiple SICStus instances is needed, then the run-times named e.g. '`sprt39_instance_01_.dll`', need to be available as well, in the same place as '`sprt39.dll`'.

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

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

If the directory containing '`sprt39.dll`' contains a directory called `sp39`, 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 ('`sp39/bin`'), as in the picture. Furthermore, the initial `library_directory/1` fact will be set to the same directory with `sp39/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
sprt39.dll
sp39\
+--- 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 '`sprt39.dll`' can be located by the DLL search.

The 39 in the file names above is derived from SICStus Prolog's major and minor version numbers, i.e. currently 3 and 9. Naming the files with version number enables applications using different SICStus versions to install the `sp*` files in the same directory.

3.3.2 Generic Runtime Systems

There are three ready-made runtime systems provided with the distributions, '`%SP_PATH%\bin\sprt.exe`',

'%SP_PATH%\bin\sprtw.exe', and '%SP_PATH%\bin\sprti.exe'. These have been created using spld:

```
% spld --main=restore '$SP_APP_DIR/main.sav' -o sprt.exe
% spld --main=restore '$SP_APP_DIR/main.sav' -i -o sprti.exe
% spld --main=restore '$SP_APP_DIR/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' (located in the same folder as the program).

The saved state is created by save_program/[1,2]. If it was created by save_program/2, the given startup goal is run. Then, user:runtime_entry(start) is run. The program exits with 0 upon normal temination and with 1 on failure or exception.

The program 'sprti.exe' assumes that the standard streams are connected to a terminal, even if they do 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 tool" in *the SICStus Prolog Manual*.

3.3.3 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 "39" for SICStus version 3.9(.x)). If SP_PATH is used, SICStus expects it to be set such that %SP_PATH%\bin contains 'sprt.sav'. See Section 3.3.1 [Runtime Systems on Target Machines], page 7.

3.4 Command Line Editing

Command line editing supporting Emacs-like commands and IBM PC 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).

3.5 The Console Window

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

In SICStus 3.8 the console was enhanced with menu access to common Prolog flags and file operations. Most of these should be self explanatory. The '`Reconsult`' item in the '`File`' menu reconsults the last file consulted with use of the '`File`' menu. It will probably be replaced in the future with something more powerful.

Note that the menus work by simulating user input to the Prolog top level or debugger. For this reason, it is recommended that the menus are only used when SICStus is waiting for a goal at the top-level (or in a break level) or when the debugger is waiting for a command.

3.5.1 Console Preferences

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 determined by the system.

`y` The Y coordinate of the top-left corner. The default is determined by the system.

On Windows 95 or 98, you will normally specify this in your '`autoexec.bat`' file. Here is an example:

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

On Windows NT and Windows 2000 or newer, you would use the '`System`' Control Panel.

Many of these settings are also accessible from the menu '`Settings`' of the console.

3.6 Windows 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/3`.
- Interruption by *C-c* only works in certain circumstances
  * *C-c* always works while Prolog code is executing. This is true for both '`sicstus.exe`' and '`spwin.exe`'.
  * *C-c* always works while '`spwin.exe`' is in a blocking read from the GUI window.
  * *C-c* will interrupt a blocking read from the (default) standard input if '`sicstus.exe`' is attached to a console window. That is, if it is started from a command prompt window.
  * *C-c* will not interrupt a blocking read from a pipe or other non-terminal. In particular, it will not interrupt a blocking read in SICStus if SICStus gets its input from a pipe, such as when running SICStus within Emacs.
  * Blocking system calls, such as those used by `library(sockets)`, are not interruptible by *C-c* in any kind of SICStus executable.
- In the windowed executable, the `user_error` stream is line buffered.
- Emacs Issues: Running under Emacs has been tried with recent versions of GNU Emacs and XEmacs. See Chapter 8 [The Emacs Interface], page 24.
  * In both GNU Emacs and XEmacs *C-c C-c* (`comint-interrupt-subprocess`) will *not* interrupt a blocking read from standard input. The interrupt will be noted

as soon as some character is sent to SICStus. The characters typed will not be discarded but will instead be used as debugger commands, sometimes leading to undesirable results.

* Choosing '`Send EOF`' from the menu, i.e. `comint-send-eof`), closes the connection to the SICStus process. This will cause SICStus to exit. This problem cannot be fixed in SICStus; it is a limitation of current versions of FSF Emacs and XEmacs (at least up to FSF Emacs 20.7 and XEmacs 21.5).

  Instead of sending and end of file, you can enter the symbol `end_of_file` followed by a period. Alternatively, a *C-z* can be generated by typing *C-q C-z*.

- Tcl/Tk: The `top_level_events` option to `tk_new/2` is not supported.
- `library(timeout)` is supported, but the time is currently measured in real time (wall-time), as opposed to process virtual time.
- `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. You should not use it as it bypasses the killed process cleanup routines.

3.7 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,pl,bas}`'
Except '`license.pl`'!

'`library\*\*.dll`'

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

4 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://dev.scriptics.com`

for MacOS X we installed Tcl/Tk (and the X windows system) using '`fink`', available at:

`http://fink.sourceforge.net`

The Tcl/Tk interface module included in SICStus Prolog 3.9 (`library(tcltk)`) is verified to work with Tcl/Tk 8.3. The current version of the interface is expected to work with version 8.1 and newer.

Under UNIX, the installation program automatically detects the Tcl/Tk version (if the user does not specify it explicitly). The distributed files are compiled for Tcl/Tk 8.3.

Under Windows, the binary distribution is compiled against Tcl/Tk 8.3. If you need to use another version of Tcl/Tk, you have to recompile `library(tcltk)`; see section "Configuring the Tcl/Tk library module under Windows" in *the SICStus Prolog FAQ*. **Please note:** You need to have the Tcl/Tk binaries accessible from your '`PATH`' environment variable, e.g. '`C:\Program Files\Tcl\bin"`'.

The GUI version of SICStus `spwin`, like all Windows non-console applications, lacks the C standard streams (`stdin`,`stdout`,`stderr`) and the Tcl command `puts` and others that use these streams will therefore give errors. The solution is to use '`sicstus.exe`' instead of '`spwin.exe`' if the standard streams are required.

4.1 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.

5 Jasper Notes

5.1 Supported Java Versions

Jasper requires at least Java 2 (a.k.a. JDK 1.2) to run. Except on Windows the full development kit, not just the JRE, is needed. **Jasper does not work with Visual J++ or Visual Café**. Unless indicated otherwise, you can download the JDK from `http://www.javasoft.com`.

Jasper is built with JDK 1.3.1 and tested with 1.2.2 and 1.3.1.

Jasper is *only* supported under the following configurations:

Solaris 2.x (SPARC and x86)

JDK 1.2
: Verified using Sun's JDK 1.2.2_06, earlier versions of 1.2 are also expected to work.

JDK 1.3
: JDK 1.3 is now supported with some limitations; see the Linux entry below.

JDK 1.3.1
: JDK 1.3.1 is now supported; see the Linux entry below. See Section 5.5 [Known Bugs and Limitations in Jasper], page 19, for when JDK 1.2 is preferred over JDK 1.3.

Linux (x86)

JDK 1.2
: Verified using Blackdown's JDK (Version 1.2.2 FCS for Linux). Downloadable from `http://www.blackdown.org/java-linux.html`.

  Sun's JDK 1.2.2 does not support native threads and therefore does *not* work.

JDK 1.3.0_02
: Other versions of JDK 1.3 are also expected to work.

  JDK 1.3 uses signals in a way that are incompatible with the way signals are used by the SICStus development system (`sicstus`). Most of the signal handlers used by (`sicstus`) are now turned off automatically before `library(jasper)` starts Java. This appears to make JDK 1.3 work with the SICStus development system. However, according to the JDK 1.3 documentation this may still cause problems. In JDK 1.3.1 the problem with conflicting uses of signals was recognized and a Java initialization option was added to reduce Java's use of signals; see the JDK 1.3.1 item below.

  Note that this is a problem only with development systems. SICStus run-time systems do not use signals, and for this reason, JDK 1.3 works e.g. when embedding SICStus in Java using the Jasper package.

JDK 1.3.1 Tested with JDK 1.3.1. JDK 1.3.1 uses signals in the same way as JDK 1.3 resulting in the same conflicts with the SICStus development system as described above. However, JDK 1.3.1 supports the option `-Xrs` which makes JDK use signals in a way that is compatible with the SICStus development system.

There are several ways to pass this flag to Java. The recommended way is to pass it with `jasper_initialize`:

```
bash> sicstus -m
...
| ?- use_module(library(jasper)),
     jasper_initialize(['-Xrs', <other options here>], JVM).
```

Alternatively, you can pass it using the (not documented in the JDK documentation) environment variable `_JAVA_OPTIONS`:

```
bash> export _JAVA_OPTIONS='-Xrs'
bash> sicstus -m
```

Using `_JAVA_OPTIONS` is currently the only way to pass this flag if you rely on the automatic Java initialization done when invoking a Java foreign resource.

Note that, also for JDK 1.3.1, this is only a problem with the SICStus development system.

Windows 95/98/NT/2000/XP
Verified using Sun's JDK 1.3.1 and JDK 1.2.2.

Other platforms
JDK 1.3 appears to be mature enough and widespread enough to make it feasible to support on other SICStus platforms as well, if required. Your input on this issue is much appreciated, especially if you are familiar with linker issues and JDK installation on the platform in question.

5.2 Getting Started

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

5.2.1 Windows

Under Windows, you should add SICStus Prolog'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\SICStus Prolog 3.9.0\bin`. For Java, it is usually '`C:\jdk1.3.1\jre\bin\hotspot`' (for JDK 1.2.2 it would be '`C:\jdk1.2.2\jre\bin\classic`').

For example (Windows NT/2000/XP):

```
set PATH=C:\jdk1.3.1\jre\bin\hotspot;%PATH%
```

```
set PATH=C:\Program Files\SICStus Prolog 3.9.0\bin;%PATH%
```

5.2.2 UNIX

When `library(jasper)` is used to embed Java in a SICStus development system or run-time system, then the run-time linker needs to be told where to find the Java libraries (e.g. '`libjvm.so`'). During installation '`InstallSICStus`' will build either the `sicstus` executable or the `jasper` foreign resource so that it contains the necessary information; the detail are platform dependent.

If you use `spld` to relink SICStus or to build a run-time system, you can use the command line option `--resource=-jasper` (note the minus sign). This tells `spld` to include the search path (*rpath*) in the executable needed to ensure that `library(jasper)` can find the Java libraries.

If you want to run `sicstus` with another Java than what was specified during installation, you can use `spld` without the `--resources` option to get a SICStus executable without any embedded Java paths. In this case, you need to set the environment variable `LD_LIBRARY_PATH` (or similar) appropriately. One example of this is to use the JDK 1.3 server version instead of the default (client) version.

Alternatively, you can use `spld` with the `--resource=-jasper` and `--with-jdk=`*DIR* options to generate a development system with embedded paths to another Java directory tree. This will only work if the alternative directory tree has the same structure as the JDK directory seen by '`InstallSICStus`'.

5.2.3 Running Java from SICStus

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 section "Jasper Library Predicates" in *the SICStus Prolog Manual*.

On UNIX, 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()` (or threads) 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
```

The '`-m`' flag is not needed, and is ignored, on Windows.

When Jasper is used in run-time systems, additional constraints apply as described in Section 3.3.1 [Runtime Systems on Target Machines], page 7. The Java to SICStus interface relies on dynamically loading the SICStus run-time system. For this reason, it is not possible to use `library(jasper)` from an executable that links statically with the SICStus run-time.

5.2.4 Running SICStus from Java

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 to specify where Java should find the Jasper native library ('`libspnative.so`' or '`spnative.dll`'), which the `SICStus` class loads into the JVM by invoking the method `System.loadLibrary("spnative")`. The `loadLibrary` 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 spnative
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)
```

On UNIX, this can be fixed by explicitly setting the Java property `java.library.path` to the location of '`libspnative.so`', like this:

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

On Windows, Java must be able to find '`spnative.dll`' through the `PATH` environment variables and setting `-Djava.library.path` can lead to problems if multiple versions of SICStus has been installed.

If this works properly, SICStus should have been loaded into the JVM address space. The only thing left is to tell SICStus where the runtime library (i.e. '`sprt.sav`') is located. On those platforms where the SICStus run-time system can determine its own location, e.g.

Windows, Solaris and Linux, the run-time system will find the runtime library automatically. Otherwise, you may choose to specify this explicitly by either giving a second argument when initializing the `SICStus` object or by specifying the property `sicstus.path`:

Example (UNIX):

```
% java -Dsicstus.path=/usr/local/lib/sicstus-3.9
```

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

If everything is set up 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" \
       -Dsicstus.path="/usr/local/lib/sicstus-3.9.0" \
       -classpath "/usr/local/lib/sicstus-3.9.0/bin/jasper.jar" \
        se.sics.jasper.SICStus
Trying to load SICStus.
If you see this message, you have successfully
initialized the SICStus Prolog engine.
```

On Windows, it would look something like this, depending on the shell used:

```
% java -classpath "C:/Program Files/SICStus Prolog 3.9.0/bin/jasper.jar" se.sics.ja
Trying to load SICStus.
If you see this message, you have successfully
initialized the SICStus Prolog engine.
```

If more than one `se.sics.jasper.SICStus` instance will be created, then the SICStus run-times named e.g. '`libsprt39_instance_01_.so`', need to be available as well. See Section 3.3.1 [Runtime Systems on Target Machines], page 7.

5.3 Jasper Package Options

The following Java system properties can be set to control some features of the Jasper package:

`se.sics.jasper.SICStus.checkSPTermAge`

This flag is unsupported.

A boolean, *true* by default. If `true`, then run-time checks are performed that attempt to detect potentially dangerous use of the `SPTerm.putXXX` family of functions. The value of this flag can be set and read with `SICStus.setShouldCheckAge()` and `SICStus.shouldCheckAge()`. This flag was *false* by default in SICStus 3.8.

The run-time checks throws an `IllegalTermException` when there is risk that a `SPTerm` is set to point to a Prolog term *strictly newer* than the `SPTerm`. In this context *strictly newer* means that there exists an open query that was opened

after the `SPTerm` object was created but before the Prolog term. See section "SPTerm and Memory" in *the SICStus Prolog Manual*, for more information.

```
java -Dse.sics.jasper.SICStus.checkSPTermAge=true ...
```

or, from Prolog:

```
jasper_initialize(
   ['-Dse.sics.jasper.SICStus.checkSPTermAge=true'],
   JVM)
```

`se.sics.jasper.SICStus.reuseTermRefs`

This flag is unsupported.

A boolean, *on* by default. If `false`, then `SPTerm.delete()` will only invalidate the `SPTerm` object, it will *not* make the Prolog side term-ref available for re-use. The value of this flag can be set and read with `SICStus.setReuseTermRefs()` and `SICStus.reuseTermRefs()`. There should be no reason to turn it *off*.

To set this flag do:

```
java -Dse.sics.jasper.SICStus.reuseTermRefs=true ...
```

or, from Prolog:

```
jasper_initialize(
   ['-Dse.sics.jasper.SICStus.reuseTermRefs=true'],
   JVM)
```

`se.sics.jasper.SICStus.debugLevel`

This flag is unsupported.

You probably should not use it in production code. It may be removed or change meaning in future releases.

An integer, zero by default. If larger than zero, then some debug info is output to `System.out`. Larger values produce more info. The value of this flag can be set and read with `SICStus.setDebugLevel()` and `SICStus.debugLevel()`.

```
java -Dse.sics.jasper.SICStus.debugLevel=1 ...
```

or, from Prolog:

```
jasper_initialize(
   ['-Dse.sics.jasper.SICStus.debugLevel=1'],
   JVM)
```

5.4 Multi threading

Some exceptions thrown in multi threaded mode may be removed in the future. The user should never catch specific exceptions, but instead catch instances of `PrologException`.

See Section 5.5 [Known Bugs and Limitations in Jasper], page 19, for details on the limitations of multi threaded Jasper.

5.5 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 by Java; they are not allowed to call native code.
- Only native threads are supported (as opposed to '`green threads`'). This is the default under Windows. Under UNIX, most JDKs use native threads per default in version 1.2.

  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-3.9.0/bin/jasper.jar \
     -Djava.library.path=[...]/lib \
     -Dsicstus.path=[...]./lib/sicstus-3.9.0 Simple

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

  Instead, do

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

  this should not be neede with JDK 1.3 or newer.

  See your JDK documentation for more info on command-line parameters to the JVM.
- Some uses of `SPTerm` will leak memory on the Prolog side. This is not really a bug but may come as a surprise to the unwary. See section "SPTerm and Memory" in *the SICStus Prolog Manual*.
- On some combinations of C-compilers and JDKs (specifically GCC with Sun's JDK), the following warning may occur:

```
% splfr simple.pl
SICStus 3.8.7 (sparc-solaris-5.7): Mon Feb 21 10:43:17 MET 2000
Licensed to SICS
{spk.ai82.c generated, 20 msec}

yes
In file included from /usr/local/jdk1.2/include/jni.h:35,
                 from spk.ai82.c:94:
/usr/local/jdk1.2/include/solaris/jni_md.h:20: warning: \
   ignoring pragma: "@(#)jni_md.h      1.11    99/02/01 SMI
```

  The warning can be safely ignored. You can suppress the warnings when using '`gcc`' by passing the options `--cflag=-Wno-unknown-pragmas` to `splfr`.
- In multi threaded mode there is a known bug which may cause a segmentation fault on the Solaris platform with JDK 1.3 We believe this to be caused by a bug in the Sun JDK that Sun has fixed in the forthcoming JDK 1.4. If you experience this problem, we recommend that you try using JDK 1.2.
- Loading multiple SICStus runtimes has not been very well tested with multi threaded Jasper.

5.6 Java Examples Directory

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

5.7 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 (http://java.sun.com/).
- JavaSoft's Java FAQ (http://java.sun.com/products/jdk/faq.html).
- JavaSoft Documentation Homepage (http://java.sun.com/docs/index.html).
- JNI Documentation (http://java.sun.com/products/jdk/1.3/docs/guide/jni/index.html).
- Yahoo's Java page (http://www.yahoo.com/Computers_and_Internet/Programming_Languages/Java/).
- The ACM student magazine *Crossroads* has published an article on the JNI (http://www.acm.org/crossroads/xrds4-2/jni.html). This article may be out of date.

6 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.
- Make the SICStus runtime DLL etc. available. See Section 3.3.1 [Run-time Systems on Target Machines], page 7. The easiest way is to put e.g. '`C:\Program Files\SICStus Prolog 3.9.0\bin`' in the '`PATH`' environment variable.

  If VB cannot find the SICStus run-time files it will report something similar to

  ```
  File not found: VBSP
  ```

- 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, on the '`PATH`' or the Windows-System directory). This is true by default if '`C:\Program Files\SICStus Prolog 3.9.0\bin`' is in the PATH environment variable, as suggested in the previous item. Note that when running your Visual Basic project in the Visual Basic debugger, then the directory of the current application is the directory that contains the Visual Basic debugger and not the directory that contains your Visual Basic project.

7 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 4.0.14. It does not work with earlier versions.

When using Berkeley DB on Windows, you should set the '`PATH`' environment variable to contain the path to '`libdb40.dll`'. Consult the Berkeley DB documentation for further info.

8 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. For information on obtaining GNU Emacs or XEmacs; see `http://www.gnu.org` and `http://www.xemacs.org`, respectively.

8.1 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-3.9.0/emacs/`' on UNIX platforms and '`C:\Program Files\SICStus Prolog 3.9.0\emacs\`' on Windows.

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

8.1.1 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 *the SICStus Prolog Manual*, for further details.

The default location for the '`info`'-files are '`<prefix>/lib/sicstus-3.9.0/doc/info/`' on UNIX platforms and '`C:\Program Files\SICStus Prolog 3.9.0\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.

9 Revision history

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

9.1 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://ftp.sics.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).

9.2 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)`.

9.3 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.

9.4 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.
- 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)`.

9.5 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.

9.6 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)`.

9.7 Changes Introduced in Version 3.8

9.7.1 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.

9.7.2 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.

9.7.3 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.

9.7.4 Generic New Features

- The `spmkds` and `spmkrs` utilities for creating stand-alone executables have been replaced by a common `spld` tool 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` tool 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 3.3 [Windows notes], page 7.
- 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 3.3.2 [Generic Runtime Systems], page 8.
- 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.

9.7.5 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.
- Changed package name from `jasper` to `se.sics.jasper`, according to JavaSoft guidelines. See Section 5.2 [Getting Started], page 15.
- Classfiles are now placed in '`jasper.jar`', which is located in `$SP_PATH/bin`. See Section 5.2 [Getting Started], page 15.
- 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 5.2 [Getting Started], page 15.

- 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.

9.7.6 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.

9.7.7 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 Windows NT
- `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; `socket_buffering/4` added
- `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 meta-calls, 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.

9.8 Changes Introduced in Version 3.8.1

Version 3.8.1 is a bugfix release only, no new features has been added.

- '`configure.in`': Removed multiple occurences of the `-n32` flag under IRIX if `cc` is used instead of `gcc`.

- 'configure.in': FreeBSD 3.x is now handled correctly.
- 'configure.in': On Linux and Solaris, SICStus is now always linked with the POSIX thread library.
- InstallSICStus: spld did not log verbose output to logfile.
- spld, splfr: Eliminated use of .. to specify relative paths. Caused problems on Windows 95/98.
- library(jasper): Green threads JDKs not supported any longer.
- library(tcltk): Tcl_FindExecutable("") is called when the 'tcltk' library is loaded, before any Tcl/Tk interpreter is created. This should fix errors related to not finding 'init.tcl' and also improve support for international character sets.
- multifile + discontiguous combination fix
- redefinition warning for multifile predicates fix
- listing/[0,1], tell/1, see/1 fixes
- avoid bogus line number info for native code
- trail compression fix
- stack_shifts (statistics/2 option) manual fix
- load_foreign_resource/1 search algorithm fix
- atom/number handling fixes
- raise error for a =.. [b|c]
- avoid SP_term_ref leaks in some functions
- prevent dangling pointer problem in displaying line number info
- check representability of compiled clauses
- prevent looping at halt and elsewhere if advice has been given
- CHR: initialization fix
- CLPFD: fixes and corrections to all_distinct/[1,2], assignment/[2,3], circuit/[1,2], serialized/[2,3], cumulative/[4,5], fdset_member/2, arithmetic
- LINDA: buffering fix

9.9 Changes Introduced in Version 3.8.2

Version 3.8.2 is a bugfix release only, no new features has been added.

- call_residue/2: fix bug when the goal called copy_term/2.
- listing/[1,2], portray_clause/[1,2], top-level: cope with constrained/attributed variables.
- portray_clause/[1,2], write_term/[2,3] with indented(true): do not juggle module prefixes.
- Foreign resources: problems with prelinked resources and with clpfd
- Foreign resources: The returned arguments from a foreign function are now properly ignored if an exception was raised with SP_raise_exception.

- Foreign resources: Added some, for now, undocumented callbacks to '`sicstus.h`'. Documented `SP_to_os`, `SP_from_os`.
- Atom garbage collector: don't reclaim undefined predicates that have pointers to them; some atom locations were not traced.
- Local stack shifter bug.
- Source info management: ensure expansion of the compiled file table.
- Backtracking from fastcode to compactcode special case.
- Bytecode relocation bug after restore.
- Compiler bug on very large clauses.
- `SP_WcxOpenHook`: incorrect prototype.
- Emulator kernel: performance bugs.
- 64-bit portability bugs.
- `library(bdb)`: a relative filename given in `db_open/5` was treated by SICStus as relative to the current working directory, but should be relative to the given BDB environment.
- `library(clpfd)`: somewhat faster arithmetic, lingering bugs in `serialized/[2,3]` and `cumulative/[4,5]`, `labeling/2` options `value/1`, `variable/1`
- The configure script did not specify the correct Irix/MIPS ABI/ISA level building with GCC.
- Added `--with=<package>` options to `spld` and `splfr` to override default installation path for third-party software packages.
- `spld`: Fixed bugs in argument handling. '`.pl`' file arguments are no longer compiled at `spld` time, but passed directly to `SP_load()`.
- `library(jasper)`: Multiple threads are allowed to call SICStus without `IllegalCallerException` being thrown. See section "Java Threads" in *the SICStus Prolog Manual*.
- `library(jasper)`: Argument-checking bug in `jasper_call_static/6` and `jasper_call_instance/6`.
- Recover properly from memory allocation failures.

9.10 Changes Introduced in Version 3.8.3

Version 3.8.3 is mainly a bugfix release. New features:

- New interface functions `SP_calloc()` and `SP_strdup()`.
- The Windows version is now up to twice as fast (measured on the benchmarks in `http://www.sics.se/sicstus/benchmarks.html`). In particular, SICStus ought to be as fast on Windows as on Linux given the same hardware. This will only affect '`pure`' Prolog code, builtins such as `assert` are not affected although the Prolog part of libraries are affected. (The change is in the byte code dispatch mechanism).
- The Windows console ('`spwin.exe`') can now save a transcript of the interaction with the Prolog top-level. The command is under the '`File`' menu. You may wish to increase the number of '`save lines`' in the '`Windows Settings`' (under the '`Settings`' menu).

- `library(clpfd)`: new constraints `disjoint1/[1,2]`, `disjoint2/[1,2]` model non-overlapping lines and rectangles.

Bug fixes:

- The Windows console: '`All Files`' should now work in file selection dialogs.
- A problem that prevented `spld` and `splfr` from working on Windows 95/98 has been fixed.
- Fixed meta-quoting of regular expressions in `spld` and `splfr`.
- `spld` warns when input files are ignored
- Runtime system executables generated using `spld` return 0 when `user:runtime_entry/1` succeeds and 1 on failure or exception.
- `SP_chdir` declares its first argument as `const char *`.
- Restore fixes for native code.
- Atom garbage collection during restore fix.
- Listing fix for disjunctions.
- Integer range manual fix.
- Avoid doing initializations twice for '`-l`' and '`-r`' files.
- Compiler fix for `once/1`.
- Buffering fix for Linda.
- Wide character handling bug fixes.
- `prolog_flag/[2,3]`: fix for runtime systems.
- `SP_unify()`: undo any bindings on failure.
- `library(bdb)`: relative filename handling fix.
- `library(clpfd)`: GC interaction, overflow detection, performance fixes.
- A problem where multiple copies of the Jasper library were loaded has been fixed. This affects all platforms. Now there is exactly one version of the Jasper shared library ('`libjasper.so`' or '`jasper.dll`').
- Jasper: `+atom` maps to `SPCanonicalAtom` instead of `SPTerm`.
- Jasper: the `+double` specifier did not work.

9.11 Changes Introduced in Version 3.8.4

Version 3.8.4 is mainly a bugfix release. New features:

- `abort/0` returns to the innermost top-level, and does not switch off the debugger.
- `library(clpfd)`: Given a term *Term* containing domain variables, `fd_copy_term(`*Term*`,`*Template*`,`*Body*`)` will compute *Template* and *Body* where *Template* is a copy of the same term with all variables renamed to new variables such that executing *Body* will post constraints equivalent to those that *Term* is attached to.
- `library(tcltk))`:

  * Added `list(CommandList)` to the possible command formats. It creates a TCL list by, in effect, calling the TCL command `list` with the result of converting each element of *CommandList*. The result is that Tcl will treat the result as a list with the same length as *CommandList* even if the elements contains spaces or other special characters.

    Current code that uses `ListOfCommands` should probably often be better off using `list(ListOfCommands)`. See the manual for details.
  * Added `writeq(Command)` and `write_canonical(Command)` as legal command specifications. Documented that `write_canonical` is the preferred way of passing Prolog terms from Prolog to Tcl and back.
  * More error checking and reporting. In particular, the output of a Prolog goal must now be in the special command format. It used to just silently generate garbage. **Potential backward compatibility issue.**

    Now the value of variables named `_` are ignored. This makes it possible to avoid errors if some uninteresting result is not in the special command format. This used to be less of a problem since such errors were silently ignored. (Note: in SICStus 3.8.5 this was changed to ignore all variables with names *starting* with underscore '_'.)
  * International (UNICODE) character now passed between Tcl/Tk and Prolog. Made the stream used internally by `library(tcltk)` always use UTF8 so that non seven bit characters gets recognized by Tcl. This transfers character codes unchanged between SICStus and Tcl so it assumes that SICStus interprets character codes as UNICODE (as this is what Tcl does).
  * `tk_num_main_windows/2` and `tk_main_window/2` no longer segfaults on Windows if `tk_new/2` has not been called. Added a "tk_new called" check to some other routines as well. The segfault occurred when, due to a bug in Tk, Tk uses stubs to access Tcl. Presently Tk uses Tcl stubs by default only on Windows.
  * The empty string resulting from an empty `CommandList` now becomes properly NUL terminated.
  * `prolog_call` now resets the FLI stack to avoid space leaks when Tcl/tk is the master and Prolog the slave.
  * Corrected some bugs in the Tcl/Tk documentation. Added examples of using the new command specifications.

- `library('linda/client')`: New predicate `shutdown_server/0`. The server keeps running after receiving this signal, until such time as all the clients have closed their connections. Courtesy of Malcolm Ryan.
- Some more options are available when the user is asked about redefining predicates.

Bug fixes:

- `skip_line/1`, `at_end_of_file/0`, `tab/2`.
- Asserting, copying or throwing terms with domain variables now raises an exception instead of crashing.
- Non-existent files and the `include/1` directive.

- GC and BDD interaction.
- `save_program/[1,2]`, `save_files/[1,2]`: check for I/O errors; problems with `'$ref'/2` terms; problems with SICStus Objects.
- `unload_foreign_resource/1`: false alarm in prelinked binaries.
- Jasper: glue code sometimes crashed when returning from a Java method that throwed an exception.
- Jasper: glue code reported errors for bogus argument numbers.
- `library(tcltk))`: Bug fixes and enhancements; see the '`New Features`' section above for details.
- `spld`/`splfr` on Windows: Errors are now properly reflected in the exit code from these programs.
- Error handling determining current directory.
- `library(clpfd)`: disequations speeded up, bugs in `disjoint1/[1,2]`, `disjoint2/[1,2]`, `element/3`, propagation, entailment detection, backward compatibility.
- Workaround for crashes when static SICStus executables, i.e. built with `spld --static`, load (non-prelinked) dynamic foreign resources. With this workaround loading a dynamic foreign resource into a static SICStus executable will still, unnecessarily, load the shared version of the SICStus runtime system ('`libsprt39.so`') but the shared runtime system will not be used. This will be fixed in a forthcoming release.

9.12 Changes Introduced in Version 3.8.5

Version 3.8.5 is mainly a bugfix release. New features:

- `copy_term/2` and `call_residue/2` now support finite domain variables.
- Representation errors due to illegal usage of finite domain variables have been replaced by more useful exceptions.
- The new exported predicate `terms:term_variables_bag/2` is like `terms:term_variables/2`, but its output argument is a list of variables in order of first occurrence.
- `bdb:db_open/5` is generalized so that a cache size can be provided.
- `clpfd:fd_neighbors/2` is a new exported predicates. It is the relation that `clpfd:fd_closure/2` is the transitive closure of.
- The Java interface has been improved; see below for new features.

Bug fixes:

- `current_atom/1` now terminates correctly.
- `once/1` is now handled correctly in ISO mode.
- `predicate_property/2` now handles built-ins correctly.
- `prolog_flag/2` alias `current_prolog_flag/2` now behave as pure relations in SICStus execution mode.
- `read/[1,2]` now handle character code 0 correctly.

- `save_files/2`, `save_predicates/2`, and `save_modules/2` do not replace given output file extensions. A '`.po`' extension will be added if none is given. Note, however, that `load_files/[1,2]` will only recognize files with a '`.po`' extension as '`.po`' files.
- `statistics(trail,L)` and `statistics(choice,L)` are more accurate.
- `stream_code/2` now handles errors correctly.
- `stream_interrupt/3` raises an existence error under Windows.
- `stream_property/2` now handles `alias/1` property for standard streams correctly.
- `stream_select/3` now returns a valid list of streams, and raises an existence error under Windows.
- The tokenizer now does not read too far ahead on non-float tokens that start like floats.
- The `s` answer to redefinition queries is now handled correctly.
- Worked around a C compiler bug affecting garbage collection under Windows (MS bug Q263609).
- A compiler bug fixed.
- Repeated restoring caused a memory leak, now sealed.
- SP_pred_refs cannot become dangling.
- Calls from C to Prolog now undergo module name expansion and goal expansion, just like calls to `call/1`.
- Bug fixes in `SP_cons_list` and `SP_cons_functor`.
- Memory overflows after `SP_open_query()` are safe.
- Stream position terms now preserve after end of stream conditions.
- Cyclic terms are detected in arithmetic function arguments.
- Predicates defined by goal expansion only can be exported.
- The `character_escapes` flag is obeyed in ISO execution mode.
- Debugger bugs fixed: showing source code in Emacs; the `a`, `o`, and `r` commands.
- The Emacs interface function `prolog-comment-region` now uses triple percent signs, to cater for `indent-region`.
- The `clpfd:full_answer` functionality has been repaired, affecting `frozen/2`, `clpfd:attribute_goal/2` and `clpfd:fd_copy_term/3`. `clpfd:fd_global/3` is now a meta-predicate.
- `sockets:socket_select/[5,6]` are now steadfast; better error handling.
- `sockets:socket_select/[5,6]` now work correctly with non-socket streams that use file descriptors on systems where sockets and file descriptors are treated the same (i.e. not Windows).
- `system:working_directory/2` is now insensitive to any loads in progress. Its arguments are not subject to `absolute_file_name/2` processing—that was never intended.
- `timeout:time_out/3` now cleans up properly after abort.
- `library(bdb)` now handles wide characters, e.g. in error messages.
- `library(clpfd)` now cleans up properly after integer overflows, and does not assume a 32-bit architecture.

- `clpfd:cumulative/[4,5]` now check that the resource limit is not exceeded by any single task.
- Glue code generated for foreign resources (C and Java) did the wrong thing for `[-term]`. The problem occurred if the foreign function did many calls to `SP_term_ref()` or if it raised an exception.
- Fixed a problem with `spld` and `splfr` on Windows 95/98.
- On AIX `spld` and `splfr` tried to use a nonexisting file. The file (‘`sprt.exp`’) is now included in the distribution.
- `halt/0` and `abort/0` are handled better in runtime systems of type `--main=load` and `--main=restore`.
- The Java interface (`library(jasper)` and `se/sics/jasper/SICStus` etc.) has been improved:
  * Fixed lots of bugs.
  * The Prolog runtime system is no longer de-initialized at random by the Java garbage collector.
  * All Java methods now properly synchronize with the Prolog runtime system to ensure thread safety.
  * There are no longer any known memory leaks when calling Java from Prolog or vice versa.
  * `SPTerm` and `SPQuery` now properly detect improper usage and raise exceptions instead of crashing in the Prolog runtime system.
  * Enhanced meta-call interface `jasper_call/4` makes foreign resources and `splfr` strictly optional when calling Java from Prolog.
  * The constructor `SPTerm()` is no longer public, it was always documented as "should really have been private". Use the contructor `SPTerm(SICStus)` instead.
  * The exception `IllegalCallerException` is no longer used. You should change your code to reflect this. One possible change is to change `throws IllegalCallerException` into `throws SPException`, this works for the 3.8.4 version as well.
  * The exception `IllegalTermException` is new. It is signalled when attempting to use a SPTerm where the corresponding term ref is no longer valid. You need to update your code (typically adding `throws IllegalTermException`). One possible change is to use the less specific `throws SPException` instead of `throws IllegalTermException`, this should work for the 3.8.4 version as well.
  * It is now possible to create terms and queries by reading from a string. See `SICStus.readFromString()` and new versions of `SICStus.openQuery()` etc.
  * `SPPredicate` is now deprecated. The preferred method is to supply module and predicate name explicitly.
  * All calls to Prolog now behave as if wrapped in `call(M:Goal)` where `M` is the module specified when creating the query. This makes goal expansion and meta argument expansion do the right thing, i.e. behave as if entered interactively.
  * It is now possible to explicitly delete a `SPTerm` object, making the Prolog side term-ref available for re-use. See `SPTerm.delete()`.

  * The documentation has been improved and expanded.
  * The examples have been updated and new examples added, notably a Swing demo with a Prolog top-level. The Prolog top-level is useful when debugging applications where Java is the top-level application.
  * `JavaServer` etc. is now more clearly marked as unsupported example code. It represents an unfinished sockets based Jasper interface. It does not belong in the `se.sics.jasper` package and will be removed at a later date.

- `library(tcltk)`. When Tcl/Tk calls Prolog, it now ignores the returned values of all unbound variables and variables with names *starting* with underscore '_'. In 3.8.4, it used to ignore only anonymous variables.

9.13 Changes introduced in version 3.8.6

Version 3.8.6 is mainly a bugfix release. New features:

- `SP_atom` documented as a data type.
- `library(jasper)` and Java foreign resources now support null object references. See section "Jasper Library Predicates" in *the SICStus Prolog Manual*.
- `library(jasper)` now works with JDK 1.3 (with some restrictions) and JDK 1.3.1 as well as JDK 1.2.2. See Section 5.1 [Supported Java Versions], page 14.
- On Linux, the path to the JDK libraries is now embedded (rpath) in the SICStus executable instead of in the Jasper foreign resource (libjasper.so). In most cases this should not be a user visible change. This was necessary due to differences between the Linux (glibc) and the Solaris dynamic loader.
- InstallSICStus should now understand TclPro directory structure when configuring Tcl/Tk.

Bug fixes:

- `splfr` code generation bug for `[-term]` and `+boolean`.
- `spld --sicstus=<PATH>` now works. You are unlikely to need it though.
- Better error reporting for incorrect type signatures to old style calls to `library(jasper)`.
- Jasper: `SPTerm.delete()` sometimes did not enable reuse of the deleted term ref.
- On some versions of Windows (NT 4 but not on Windows 2000) Java would crash under the following circumstances. '`java.exe`' is the main application, Java tries to do `use_module(library(jasper))`, the *short* pathname of the '`SICStus Prolog\bin`' folder is on the `PATH` environment variable. As a work-around for the underlying Win32 `LoadLibrary` bug SICStus will now always use the long pathname when loading foreign resources.
- On Windows, you can sometimes get an floating point divide by zero exception when embeding SICStus into other applications. The symptom was a crash with something like `"Exception: 0xc000008e (EXCEPTION_FLT_DIVIDE_BY_ZERO)"`. This also happened for some applications that use Visual Basic for Applications (VBA) with the

SICStus Visual Basic module (vbsp.dll). A thorough discussion of this issue and a solution is available in '`library/vbsp/sp_fpwrap.h`'. This solution is now used by the SICStus Visual Basic module. We have had reports of this issue affecting FileMaker, Rational Rose and Visio.

- `library(bdb)` now complains if run with a different version of Berkeley DB than what was used for building it (BDB 2.7.7).
- Now uses `malloc()` for memory allocation when invoking SICStus from Java, also on Linux. The default memory allocation method (using `sbrk`/`brk`) is not thread safe on any platform.
- Avoid spurious error message after `C-c a`.
- Loading '`.po`' files and saved states: work around GCC bug affecting endianness conversion.
- Standard term comparison on stream position terms didn't work after `seek/4`.
- `dif/2` and friends: memory management bug.
- `format/[2,3]`: avoid spurious time-out exceptions.
- `freeze(V,V)` behavior bug.
- `number_chars/2, number_codes/2`: bug affecting empty lists.
- `library(bdb)` now verifies that BDB version 2.7.7 is used.
- `library(clpfd)`: missing distribution files; buggy action handling of user-defined global constraints; wrong answers in `disjoint1/[1,2]`, `disjoint2/[1,2]`, `serialized/[2,3]`, and `cumulative/[4,5]`; integer overflow checks; error detection in FD set operations; complexity of `fd_closure/2` and `fd_copy_term/3`; entailment action in `element/3`.
- `library(clpq,clpr)`: bug affecting `bb_inf/3` and strict inequalities.
- `library(jasper)` now enforces the use of `malloc` for memory management, also on Linux.
- `sockets:socket_buffering/4`: bug handling 3rd arg.
- `library(tcltk)`: some demos depended on current working directory.
- `library(xref)`: handling of `catch/3`.

9.14 Changes introduced in version 3.8.7

Version 3.8.7 is mainly a bugfix release. New features:

Support for MacOS X 10.x. This includes support for Tcl/Tk (see Chapter 4 [Tcl/Tk Notes], page 13).

Made the Windows version faster using an improved version of the tweaks used in 3.8.3.

Bugs fixed:

Performance problems in metacalls, exception handling, `findall/3` and friends, `sort/2`, `keysort/2`.

`atom_codes/2`, `atom_chars/2`, `number_codes/2`, and `number_chars/2` were not steadfast.

`library(timeout)`: Some timeouts were ignored if they occurred during exception handling.

`library(timeout)`: If the timer cannot be set up (using `setitimer`), then a system error exception is raised. This happens on Solaris in multi-threaded applications, e.g. when using SICStus with Java.

`SP_deinitialize` not working properly.

Infinite loop in dynamic code memory management.

Arguments to format spec '`~@`' were not called as fully general goals.

Loading and unloading resources was vulnerable to changing file search paths.

`listing/[0,1]` did not module prefix bodies of imported predicates correctly.

`library(clpfd)`: some Boolean constraints were incorrectly macro-expanded; some type errors merely failed; over-zealous integer overflow detection in arithmetic.

`sockets:socket_select/[5,6]` will work correctly also for input buffered socket streams. You no longer need to disable buffering with `sockets:socket_buffering/4` just to get `sockets:socket_select/[5,6]` to work.

`library(sockets)`: A signal delivered to the process will no longer cause any socket predicates to give an error (proper '`EINTR`' handling). A downside to this is that SICStus no longer is interruptible with `^C` ('`SIGINT`') while blocking except in `sockets:socket_select/[5,6]`. In the case of `sockets:socket_select/[5,6]`, signal delivery will be treated as if the `select()` call had a zero timeout. Such spurious timeouts are a **potential backward compatibility issue.**

`sockets:socket_buffering/4`: A crashing bug introduced in 3.8.6. Added argument validation so that it now fails on non-socket streams and other incorrect arguments. Note that the primary reason for changing socket buffering prior to 3.8.7 is gone since `sockets:socket_select/[5,6]` now work also for input buffered sockets.

`system:file_property/2` failed to recognize symbolic links as such.

Jasper: A Prolog side memory leak in `SPTerm.consFunctor`. This also affected all read-from-string type methods.

Jasper: `SPTerm.getNumberChars` called an undefined native method.

On UNIX, the Prolog run-time would temporarily unblock all OS signals. This could lead to unexpected behavior when linking SICStus with other applications that use signals (such as Java).

`library('clpqr/examples/mip')` would not load properly.

Solaris, `spld --static` used incorrect options when specifying static linking to Sun Workshop C-compiler.

9.15 Changes Introduced in Version 3.9

9.15.1 Message and Query System

Every message issued by the Prolog system is displayed using the built-in predicate `print_message/2`. Similarly, all user input is handled by the new built-in predicate `ask_query/4`. The behavior of these predicates can be customized by a number of new hooks. For example, all I/O can be redirected from the standard streams to Tcl/Tk or similar windows. The default appearance of system messages has also changed.

The mapping from Prolog terms representing messages to printed messages is now completely transparent and customizable. The new `library('SU_messages')` file defines the mapping rules. Runtime systems can display system messages in standard format by loading this file.

A number of new hooks have been added for the message and query system: `generate_message_hook/3`, `generate_message/3`, `message_hook/3`, `print_message_lines/3`, `query_hook/6`, `query_class_hook/5`, `query_input_hook/3`, `query_map_hook/4`.

See section "Messages and Queries" in *the SICStus Prolog Manual*.

9.15.2 Determinacy Checker

The determinacy checker, `spdet`, is a new, powerful tool originally written by Dave Bowen and Peter Schachte.

The determinacy checker can help you spot unwanted nondeterminacy in your programs. This tool examines your program source code and points out places where unintended nondeterminacy may arise. Unintended nondeterminacy should be eradicated because

1. it may give you wrong answers on backtracking
2. it may cause a lot of memory to be wasted

See section "The Determinacy Checker" in *the SICStus Prolog Manual*.

9.15.3 Cross-Referencer

`library(xref)` of previous releases has been replaced by a much more powerful tool, `spxref`, originally written by Tom Howland.

The main purpose of the cross-referencer is to find unreachable code. To this end, it begins by looking for a definition for `user:runtime_entry/1` and also looks for initializations, hooks, module export lists and `public` directives to start tracing the reachable code from.

A second function is to aid in the formation of module statements. It can list all of the required `module/2` and `use_module/2` statements by file.

See section "The Cross-Referencer" in *the SICStus Prolog Manual*.

9.15.4 Common Object Model Client

On Windows, `library(comclient)` makes it possible to control COM Automation objects. See section "Com Client" in *the SICStus Prolog Manual*.

9.15.5 The PiLLoW Web Programming Library

The PiLLoW library is a free Internet/WWW programming library which simplifies the process of writing applications for such environment. The library provides facilities for generating HTML or XML structured documents by handling them as Prolog terms, producing HTML forms, writing form handlers, processing HTML templates, accessing and parsing WWW documents (either HTML or XML), accessing code posted at HTTP addresses, etc. See section "PiLLoW" in *the SICStus Prolog Manual*.

9.15.6 New CLPFD Features

Generally, performance has been improved, in terms of CPU time as well as memory. Specific new features:

- `labeling/2` takes a new option which supports limited discrepancy search. To support this, the `apply_bound/1` API has been replaced by predicates called `first_bound/2` and `later_bound/2`.
- The new constraint `knapsack/3` is a domain consistent special case of `scalar_product/4`.
- The new constraint `global_cardinality/2` constrains the number of occurrences of given integers in a list.
- Arbitrary N-ary relations can be defined with the new constraint `case/4`.
- `disjoint/2` takes a new option which tells the constraint to perform stronger reasoning if some rectangles have the same origin and certain other conditions are fulfilled.
- `serialized/3` and `cumulative/5` take a new option which tells the constraint whether to only adjust domain bounds. This is now the default.
- The `precedences/1` option of the same constraints has a more general format.
- The new constraint `cumulatives/[2,3]` generalizes `cumulative/[4,5]` by considering multiple resources, positive and negative resource consumption, and lower and upper bounds.

9.15.7 All-in-one Executables

It is now possible to embed saved states into an executable. Together with static linking, this gives an executable which does not depend on external SICStus files. See section "All-in-one Executables" in *the SICStus Prolog Manual*.

9.15.8 Multiple SICStus Run-Times in a Process

It is now possible to have more than one SICStus run-time in a single process. See section "Multiple SICStus Run-Times" in *the SICStus Prolog Manual*, for details.

9.15.9 Miscellaneous

- The new built-ins `read_line/[1,2]` read one line of input into a list of character codes.
- The new built-in `stream_position_data/3` accesses the fields of a stream position term.
- The new built-in `goal_source_info/3` decomposes annotated goals into a goal proper and a source position. Mostly used in displaying error messages.
- The built-in `undo/1`, which posts a goal for execution on backtracking, has been revived.
- `absolute_file_name/3` is a new variant of `absolute_file_name/2` taking a number of options to give full control over the conversion from relative to absolute filename.
- `read_term/[2,3]` takes a new option which controls whether the *layout-text-item* which follows the terminating `.` should be consumed.
- A resource error exception is raised when there is insufficient memory to continue execution. In previous versions, this condition caused SICStus Prolog to abort the computation and to reinitialize itself.
- `trimcore/0` trims the Prolog stacks.
- Debugger changes:
  * Breakpoint conditions now can contain the usual Prolog connectives '`,`', '`;`', '`->`', and '`\+`'.
  * A breakpoint for which the test conditions evaluated successfully is always applied, even if the action conditions fail.
  * A hook, `user:breakpoint_expansion/2`, has been added for user defined breakpoint conditions.
  * Advice-points are now allowed to set all three action variables (mode, show, command).
- `library(bdb)` provides a way to flush all modified records to disk after each operation.
- `library(charsio)` provides the new predicates `read_term_from_chars/3` and `write_term_to_chars/[3,4]`.
- `library(system)` provides the new predicates `now/1` and `datime/2`.
- `library(clpq,clpr)` provides the new predicate `projecting_assert/1`.
- `SP_set_read_hook()` is now obsolescent. See section "Hooks" in *the SICStus Prolog Manual*.
- `SP_event()` can now be called from arbitrary OS threads. See section "Calling Prolog Asynchronously" in *the SICStus Prolog Manual*.
- `splfr` generates a header file from the `foreign/[2,3]` declarations. This file should be included in the corresponding C file to protect against incorrect foreign declarations and also to ensure compatibility with various compile time settings used by `splfr`.

The name of the file can be specified with `--header=`*NAME*. To produce the file and nothing more, you can use the new `--nocompile` flag as in

```
splfr --header=foo.h --nocompile foo.pl
```

This is especially useful when '`foo.h`' is a Makefile prerequisite.

- `splfr` and `spld` now use descriptive names for generated files, especially when the flag `--keep` is specified.
- On Windows, it is now possible to link an application with a static version of the SICStus run-time. It is also possible to link with static versions of foreign resources.
- A statically linked SICStus run-time can load foreign resources dynamically.
- `spld --moveable` now produces an executable which can be moved, together with its directory tree (see Section 3.3.1 [Runtime Systems on Target Machines], page 7) on Solaris and Linux (it always worked on Windows).
- New `spld` options,

  `--lang` Selects the Prolog dialect used for run-time systems created with `--main=load` and `--main=restore`.

  `--memhook`
  Selects memory allocation method (`SP_set_memalloc_hooks`).

  You can use `spld --help` to get details on these and other options.
- The development system (`sicstus` and, on Windows, `spwin`) now take option `--iso` and `--sicstus` to start up in ISO Prolog mode and SICStus Prolog mode respectively.
- New C preprocessor macros with version info are now defined when including '`sicstus.h`'. For SICStus 3.9.0 they are defined as:

```
#define SICSTUS_MAJOR_VERSION 3
#define SICSTUS_MINOR_VERSION 9
#define SICSTUS_REVISION_VERSION 0
#define SICSTUS_BETA_VERSION 0

/* Two digit position for MAJOR, MINOR and REVISION */
#define SICSTUS_VERSION 30900
```

- New C API function `SP_read_from_string` takes a string representation of a Prolog term and a table of variable values and creates a Prolog term. By reading a goal from a string and passing the resulting term to `call/1`, this also gives a very simple way to call arbitrary goals from C. See section "Mixing C and Prolog" in *the SICStus Prolog Manual*.
- New C API function `SP_define_c_predicate` makes it possible to dynamically define a Prolog predicate that calls a C function. See section "Mixing C and Prolog" in *the SICStus Prolog Manual*.
- New C API functions `SP_mutex_lock`, `SP_mutex_unlock`, C type `SP_mutex` and static initializer `SP_MUTEX_INITIALIZER`. These provide a platform independant (recursive) mutual exclusion lock. These are mainly useful when more than one SICStus run-time is used (in different threads) in the same process. See section "Operating System Services" in *the SICStus Prolog Manual*.

- The environment variables `SP_APP_DIR` and `SP_RT_DIR` are set (by `SP_initialize`) to the folder containing the executable and the folder containing the SICStus run-time, respectively. This provides a location independent way to locate files (such as saved states) that are located together with the application. Also available as the file search aliases `application` and `runtime`. See section "Input Output" in *the SICStus Prolog Manual*.
- On Win32, the C signal `SIGBREAK` will cause `halt/0` to be called in a development system. This will ensure that `halt/0` is called in '`sicstus.exe`' when the console window is closed or when the user logs off. The windowed version of SICStus ('`spwin.exe`') also sends itself a `SIGBREAK` when the GUI window is closed, causing `halt/0` to be called.

9.15.10 Incompatibilities with Previous Versions

- The semantics of `absolute_file_name/2` has changed slightly. It no longer looks for an existing file with a '`.pl`' extension if the given relative file name has no extension. The old behavior can be achieved with:

```
old_absolute_file_name(RelFileSpec, AbsFileName) :-
   Options = [ file_type(source),
               access(exist),
               file_errors(fail) ],
   (  absolute_file_name(RelFileSpec, AbsFileName, Options)
   -> true
   ;  absolute_file_name(RelFileSpec, AbsFileName, [])
   ).
```

- The hook `debugger_command_hook/2` takes a different first argument.
- The built-in `print_message/2` does not take a `force/1` severity.
- The UNIX-specific built-ins `stream_interrupt/3` and `stream_select/3` were unsafe, and have been removed. The latter can be replaced by the following equivalent definition, which works for file descriptor streams on UNIX and, for socket streams, on Windows as well:

```
:- use_module(library(sockets)).
stream_select(Streams, TimeOut, ReadStream) :-
        socket_select([], _, TimeOut, Streams, ReadStream).
```

- Exceptions in hooks are now caught locally instead of propagating into the calling code.
- The Prolog predicates for generating glue code for the foreign language interface are no longer available. The only supported method for generating foreign resources are `splfr`. The removed predicates are `link_foreign_resource/6`, are `prepare_resource_table/2`, and `prepare_foreign_resource/3`. See section "The Foreign Resource Linker" in *the SICStus Prolog Manual*, for details on using `splfr`.
- The predicate `reinitialise/0` is gone.
- The `--import` flag to `splfr` is no longer supported.
- The CLPFD `apply_bound/1` API for branch & bound search has been replaced by predicates called `first_bound/2` and `later_bound/2`.

- The CLPFD constraints `all_different/[1,2]` and `all_distinct/[1,2]` require bounded domains.
- The CLPFD constraints `serialized/[2,3]` and `cumulative/[4,5]` only prune the bounds of domains, not inside them, unless the `bounds_only(false)` option is given.
- The Berkeley DB interface is now compatible with Berkeley DB 4.0.14 instead of 2.7.7.
- `library(db)` has been removed.
- `library(xref)` has been replaced.
- `library(flinkage)` is no longer supported. It is still present to help porting really old code.
- The header file '`<sicstus/sicstus.h>`' now uses stricter function type prototypes. This may expose problems in code that used to compile without warnings in previous versions of SICStus.
- Java foreign resources are no longer supported. Java foreign resources offer no advantages compared to `jasper_call/4` introduced in SICStus 3.8.5 so you should migrate your code even if you are still using SICStus 3.8.
- Java now loads '`libspnative.so`' ('`spnative.dll`' on Windows) where it used to load '`libjasper.so`'. This will probably be invisible to most users.
- The Java methods in `se.sics.jasper` that operate on a SICStus object will now throw an error if called from a thread other than that which created the SICStus object. This is similar to the behavior in early versions of 3.8.

  SICStus 3.9 introduces a new thread safe Jasper API, which is not backwards compatible with the Jasper API of 3.8. However, the old API has been amended to be compatible with the thread safe API, making it possible to write Java programs that can run in both modes.
- Signal handlers installed by `SP_signal` are now deferred until Prolog can call them safely. This makes `SP_signal` unsuitable for handling synchronous signals such as '`SIGSEGV`'. See section "Signal Handling" in *the SICStus Prolog Manual*.

9.15.11 Bugs Fixed

- `at_end_of_stream/0`, `get/1`, `get0/1`, `put/1`, `skip/1`, `tab/1` incorrectly required that the current input stream be a text stream.
- `stream_property(S, end_of_stream(...))` now works for tty streams.
- `user:goal_expansion/3` was called with the wrong second argument for imported and built-in predicates.
- `require/1` was broken.
- The escape sequence '`\c`' could be misread.
- In ISO mode, terms were printed with non-ISO escape sequences.
- `charsio:format_to_chars/[3,4]` are now meta-predicates.
- `charsio:format_to_chars/[3,4]` is now re-entrant.
- `library(clpfd)`: some Boolean constraints were incorrectly macro-expanded; some type errors merely failed; over-zealous integer overflow detection in arithmetic; unifying domain variables did not work reliably.

- `library(clpq,clpr)`: Variables introduced via `ordering/1` were not initialized fully.
- `library('linda/client')`: `shutdown_server/0` now raises an existence error if there is no connection to the server available.
- Emacs line break points now work for module files.
- Windows now always uses `^Z` where UNIX uses `^D` to signal end of file on input from the terminal.
- On Windows, when running SICStus under Emacs, the SICStus process sometimes got stuck in a tight loop when Emacs closed the connection to the SICStus sub-process. It will now exit instead; see Section 3.6 [Windows limitations], page 11.
- On Windows, `splfr` now converts the resource name to lowercase. This is to match the fact that SICStus converts path names to lowercase on Windows.
- As of SICStus 3.8.7, for `sockets:socket_select/[5,6]`, signal delivery will be treated as if the `select()` call had a zero timeout. Such spurious timeouts are now handled invisibly within `library(sockets)` and will not be seen by user code.

9.15.12 Known Problems with This Release

- Some Prolog libraries that depend on foreign code (e.g. `library(bdb)`, `library(tcltk)`) can only be loaded by one SICStus run-time in the same process. The second run-time that attempts to load e.g. `library(bdb)`, will raise an exception. This only affects code that loads several SICStus run-times (e.g. using the Java interface).
- The API for associating destructors with external objects is in place but is not documented. It is, at present, only used by the COM client library. Contact `sicstus-support@sics.se` if you have any questions about it.
- See Section 9.15.10 [3.9 Incompatibilities], page 49.

10 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. An experimental workaround for the Linux 128 MB limit is available from `sicstus-support@sics.se`.

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

The number of atoms created may not exceed 262143 (33554431) on 32-bit (64-bit) architectures.

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.

11 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

Report bugs through the web interface

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

or 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