

SICStus Prolog Release Notes

by the Intelligent Systems Laboratory

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

Release 4.0.0
March 2007

Swedish Institute of Computer Science

sicstus-request@sics.se

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

Copyright © 1995-2007 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.

Table of Contents

1	Overview	1
2	Platforms	2
3	Release Notes and Installation Guide for UNIX	3
3.1	Installation	3
3.1.1	Prerequisites	3
3.1.1.1	C Compiler and Linker	3
3.1.2	The Installation Script	3
3.1.3	The Uninstallation Script.....	4
3.2	Platform Specific Notes	4
4	Release Notes and Installation Guide for Windows	6
4.1	Requirements	6
4.2	Installation	6
4.3	Windows Notes	7
4.4	Command Line Editing	7
4.5	The Console Window	8
4.5.1	Console Preferences	8
4.6	Windows Limitations	9
5	Tcl/Tk Notes	10
6	Berkeley DB Notes	11
7	The Emacs Interface	12
7.1	Installation	12
7.1.1	Installing On-Line Documentation	12
8	Revision History	13
8.1	What Is New In Release 4	13
8.1.1	Virtual Machine	13
8.1.2	Prolog Language.....	13
8.1.2.1	Single Language Mode	13
8.1.2.2	DCG Notation	13
8.1.2.3	Asserting Terms with Attributed Variables	13
8.1.2.4	Arithmetic.....	14
8.1.2.5	Syntax	14

8.1.2.6	Prolog Flags	14
8.1.2.7	Stream Properties	14
8.1.2.8	Statistics Keywords	14
8.1.2.9	Built-In Predicates	14
8.1.2.10	Hook Predicates	17
8.1.3	Library Modules	18
8.1.4	Input-Output System	21
8.1.5	Foreign Language APIs	22
8.1.5.1	Foreign Language Interface	22
8.1.5.2	C API Functions	22
8.1.5.3	Java API	23
8.2	Guide to Porting Code from Release 3	23
8.3	Limitations in the Current Release	25
9	Generic Limitations	27
10	Contact Information	28

1 Overview

These notes summarize the changes in release 4 wrt. previous SICStus Prolog releases as well as changes introduced by minor releases and their patch releases. Platform specific information pertaining to certain parts of the system are also documented herein.

2 Platforms

Binary distributions of Release 4.0 are available for the following platforms. Additional platforms are available. If your platform is not listed, please let us know (sicstus-request@sics.se).

Intel/x86, Windows 2000/XP/Vista, 32 bits

Intel/x86, Linux glibc 2.3, 32/64 bits

Built and tested on Red Hat Enterprise Linux 4

Intel/x86, Linux glibc 2.2, 32 bits

Built and tested on Red Hat Linux 7.2

Intel/x86, Solaris 10, 32/64 bits

Intel/x86, Mac OS X 10.4, 32/64 bits

PowerPC, Mac OS X 10.4, 32 bits

PowerPC, AIX 5.1L, 32/64 bits

Sparc, Solaris 8, 32/64 bits

3 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 3.1 \[UNIX installation\]](#), page 3. For example:

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

3.1 Installation

Installation of SICStus under UNIX is performed by an installation (Shell) script `InstallSICStus`, which interacts with the user to obtain options such as where to install SICStus. The Java based *SICStus Installer Tool* is a graphical front-end to the installation script, which automates downloading and installation. The SICStus Installer Tool is available from the download page. Use of the SICStus Installer Tool is strictly optional but may be convenient, especially on platforms such as Mac OS X, that, by default, lack C compiler.

3.1.1 Prerequisites

3.1.1.1 C Compiler and Linker

A full SICStus installation requires a C compiler and a linker to perform final link steps on the installation machine.

If a C compiler is not available, it is possible to use a *pre-built installation* on some platforms.

Pre-built installation is only recommended as a last resort; it is available from the SICStus Installer Tool or by invoking `InstallSICStus` with the `'--all-questions'` argument.

A disadvantage with the pre-built installation is that SICStus libraries that interface to third-party products (Tcl/Tk, Berkeley DB, Java) may not work, or may require environment variables such as `LD_LIBRARY_PATH` to be set. Another disadvantage is that `spld` and `splfr` may not work unless you manually adjust the `spld` configure file. Of course, neither `spld` nor `splfr` will work anyway if you do not have a C compiler.

3.1.2 The Installation Script

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 an interactive installation script called `InstallSICStus`. Type

```
% ./InstallSICStus
```

and follow the instructions on the screen. As an alternative, the SICStus Installer Tool can be used to download the SICStus files and invoke the installation script.

During 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)` and `library(tcltk)`). This is done in order to adapt to local variations in installation paths and versions.

Invoke `InstallSICStus` with the `'--help'` argument to get a list of options.

Compiling `SICStus` from the sources requires a source code distribution. Contact `sicstus-support@sics.se` for more info.

3.1.3 The Uninstallation Script

To uninstall `SICStus` the script `UnInstallSICStus` can be run. It is created during installation in the same directory as `InstallSICStus`.

3.2 Platform Specific Notes

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

Solaris SPARC 64-bit

You cannot install (or build) the 64 bit version of `SICStus` using `gcc 2.x`. You need to use the Sun Workshop/Forte compiler, version 5.0 or later. `InstallSICStus` will try to find it during installation but if that fails, you can set the environment variable `CC` to e.g. `'/opt/SUNWspro/bin/cc'` before invoking `InstallSICStus`. Using `gcc 3.x` does seem to work but has not yet received much testing. To install with `gcc 3.x`, set the environment variable `CC` appropriately before invoking `InstallSICStus`.

The following libraries are not supported: `library(bdb)`, `library(tcltk)`.

Solaris 8 `Library timeout` does not work with the default thread implementation in Solaris 8. Instead the “Alternate Thread Library (T2)” must be used. This is ensured automatically for executables built with the `spld` tool. See http://developers.sun.com/solaris/articles/alt_thread_lib.html for further information.

This problem does not affect Solaris 9 or later.

Mac OS X

An executable built with `spld` will only work if there is a properly configured subdirectory `'sp-4.0.0'` in the same directory as the executable; see [section “Runtime Systems on UNIX Target Machines” in *the SICStus Prolog Manual*](#). Alternatively, the option `'--wrapper'` can be passed to `spld`. In this case a wrapper script is created that will set up various environment variables and invoke the real executable.

When using third-party products like `BDB`, you may need to set up `DYLD_LIBRARY_PATH` so that the Mac OS X dynamic linker can find them. When

using the SICStus development executable (`sicstus`), a wrapper script does this automatically.

Sometimes, the default limit on the process's data-segment is unreasonably small, which may lead to unexpected memory allocation failures. To check this limit, do

```
tcsch> limit data
datasize 6144 kbytes
bash> ulimit -d
6144
```

This indicates that the maximum size of the data-segment is only 6 Mb. To remove the limit, do

```
tcsch> limit datasize unlimited
datasize unlimited
bash> ulimit -d unlimited
bash> ulimit -d
unlimited
```

Please note: `limit` (`ulimit`) is a shell built-in in `csch/tcsch` (`sh/bash`). It may have a different name in other shells.

Please note: The limit will also affect SICStus when started from within Emacs, e.g. with `M-x run-prolog`. To change the limit used by Emacs and its sub-processes (such as SICStus) you will need to change the limit in the shell used to start Emacs. Alternatively you can create a shell wrapper for the `emacs` command.

`library(timeout)` does not work reliably when Mac OS X runs on a multi-CPU machine. In particular, timeouts tend to happen much later than they should. This is caused by an OS bug. One workaround is to disable all but one CPU using the "Processor" control in the "System Preferences" or the `hwprefs` command. These utilities are part of "CHUD" which can be installed as part of Apple XCode. The underlying bug is related to `setitimer(ITIMER_VIRTUAL)` and has been observed at least up to Mac OS X 10.4.8 (Darwin 8.8.1).

File names are encoded in UTF-8 under Mac OS X. This is handled correctly by SICStus.

If SICStus encounters a file name that is not encoded in UTF-8, it will silently ignore the file or directory. This can happen on file systems where files have been created by some other OS than Mac OS X, e.g. on network file servers accessed by other UNIX flavors or Windows.

The default character encoding for the SICStus standard streams is based on the current locale which is POSIX/C, i.e. US ASCII, by default on Mac OS X. This will come in conflict with the default character encoding for the Terminal application which is UTF-8. A clickable launcher for SICStus is optionally installed in the Applications folder. This launcher will set the character encoding of the standard streams to UTF-8 for both the Terminal and SICStus.

4 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 4.0.0`). Here, `%SP_PATH%` is just a place-holder; you usually do not need to set the environment variable `SP_PATH`, but see [section “CPL Notes” in the SICStus Prolog Manual](#). For example:

```
C:\> set PATH=C:\Program Files\SICStus Prolog 4.0.0\bin;%PATH%
```

To use `splfr` and `spld`, you must also include Microsoft Visual Studio (or at least its C compiler and linker). The easiest way is to run `vsvars32.bat` from the Visual Studio distribution.

To use the respective library modules, you must also include the paths to Tcl/Tk (see [Chapter 5 \[Tcl/Tk Notes\], page 10](#)) and Berkeley DB (see [Chapter 6 \[Berkeley DB Notes\], page 11](#)) onto the `PATH` environment variable if the installer for Berkeley DB and Tcl/Tk have not done so already.

4.1 Requirements

- Operating environment: Microsoft Windows 2000 SP4, XP SP2 or Vista (including x64 but not IA64 versions of XP and Vista). Windows XP or later is recommended.
- Available hard drive space: 200 Mbytes (approximate)
- For interfacing with C or C++, or for using `spld` or `splfr`: C compiler and related tools from Microsoft Visual Studio 2005 SP1 (a.k.a. VS 8).

Microsoft offers free editions of its C compilers. It is probably possible to make these work as well but they may require other tools or downloads.

4.2 Installation

The development system comes in two flavors:

1. A console-based executable suitable to run from a DOS-prompt, from batch files, or under Emacs. See [Section 4.4 \[Command Line Editing\], page 7](#).
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 by starting sicstus from the Start menu (`spwin.exe`) and selecting **Enter Licence** from the **Settings** menu. Entering the license may require Administrative rights. Running SICStus should be possible from a limited account.

4.3 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.
- 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 4.0.0\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/>.
- We recommend that SICStus be installed by a user with administrative privileges and that the installation is made ‘For All Users’.

If SICStus is installed for a single user, SICStus will not find the license information when started by another user. In this case, the windowed version of SICStus (`spwin`) will put up a dialog where a license can be entered.

4.4 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:

<code>^h</code>	erase previous char
<code>^d</code>	erase next char
<code>^u</code>	kill line
<code>^f</code>	forward char
<code>^b</code>	backward char
<code>^a</code>	begin of line
<code>^e</code>	end of line
<code>^p</code>	previous line
<code>^n</code>	next line
<code>^i</code>	insert space
<code>^s</code>	forward search
<code>^r</code>	reverse search

<code>^v</code>	view history
<code>^q</code>	input next char blindly
<code>^k</code>	kill to end of line

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

Option Value

on separate lines. Recognized options are:

<code>lines</code>	<i>Value</i> is the number of lines in the history buffer. 1-100 is accepted; the default is 30.
<code>save</code>	<i>Value</i> is either 0 (don't save or restore history buffer) or 1 (save history buffer in <code>‘%HOME%\spcmd.hst’</code> 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).

4.5 The Console Window

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

The console comes with a 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. The console 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 only be 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.

4.5.1 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 the following:

<code>sl</code>	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.
<code>rows</code>	The initial number of lines. The default is 24.
<code>cols</code>	The initial number of columns. The default is 80.
<code>x</code>	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.

Many of these settings are also accessible from the menu ‘Settings’ of the console.

4.6 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 ‘\\’.
- All file names and paths are normalized when expanded by `absolute_file_name/3`. This is to simulate the case insensitivity used by Windows file systems. This means that files created by SICStus may have names on disk that differs in case from what was specified when the file was created.
- Emacs Issues: Running under Emacs has been tried with recent versions of GNU Emacs and XEmacs. See [Chapter 7 \[The Emacs Interface\], page 12](#).
 - 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`.
- Under Windows, `statistics(runtime, ...)` measures user time of the thread running SICStus (the main thread) instead of process user time. This makes `statistics(runtime, ...)` meaningful also in a multi-threaded program.

5 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://tcl.sourceforge.net>

n A better alternative may be to use one of the free installers available from:

<http://www.activestate.com>

SICStus for Mac OS X uses Aqua Tcl/Tk. The Aqua version of Tcl/Tk uses the native Aqua user interface. Mac OS 10.4 includes Aqua Tcl/Tk. For earlier versions of Mac OS X, Aqua Tcl/Tk can be downloaded, for free, from:

http://www.apple.com/downloads/macosx/unix_open_source/

The Tcl/Tk interface module included in SICStus Prolog 4.0.0 (`library(tcltk)`) is verified to work with Tcl/Tk 8.4. 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). Except as noted above, the distributed files are compiled for Tcl/Tk 8.4.

Under Windows, the binary distribution is compiled against Tcl/Tk 8.4. 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` instead of `spwin` if the standard streams are required.

6 Berkeley DB Notes

`library(bdb)` is built on top of Berkeley DB. Berkeley DB can be downloaded from:

<http://www.oracle.com/database/berkeley-db>

Berkeley DB for Mac OS X can be installed using MacPorts
<http://trac.macosforge.org/projects/macports/wiki>.

`library(bdb)` is built using version 4.5.20. It may be possible to recompile it to work with other versions as well.

When using Berkeley DB under Windows, you should set the `PATH` environment variable to contain the path to `'libdb45.dll'`. Consult the Berkeley DB documentation for further info.

7 The Emacs Interface

The Emacs Interface was originally developed for GNU Emacs 19.34 and is presently being maintained using XEmacs 21.1 and tested with GNU Emacs 21.2. For best performance and compatibility and to enable all features we recommend that the latest versions of GNU Emacs or XEmacs be used. For information on obtaining GNU Emacs or XEmacs; see <http://www.gnu.org/software/emacs/> and <http://www.xemacs.org>, respectively.

7.1 Installation

The Emacs interface is distributed with SICStus and installed by default. The default installation location for the Emacs files is '`<prefix>/lib/sicstus-4.0.0/emacs/`' on UNIX platforms and '`C:\Program Files\SICStus Prolog 4.0.0\emacs\`' under Windows.

For maximum performance the Emacs Lisp files (extension '`.el`') should be compiled. This, completely optional step, 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.

The easiest way to configure the Emacs interface is to load the file '`sicstus_emacs_init.el`' from your '`.emacs`' file. It will find the SICStus executable and do all initialization needed to use the SICStus Emacs interface.

7.1.1 Installing On-Line Documentation

It is possible to look up the documentation for any built in or library predicate from within Emacs (using `C-c ?` or the menu). For this to work Emacs must be told about the location of the '`info`'-files that make up the documentation.

If you load the file '`sicstus_emacs_init.el`' from your '`.emacs`' file then Emacs should be able to find the SICStus documentation automatically; see [section "Installation" in the SICStus Prolog Manual](#), for further details.

8 Revision History

This chapter summarizes the changes in release 4 wrt. previous SICStus Prolog releases as well as changes introduced by patch releases.

8.1 What Is New In Release 4

8.1.1 Virtual Machine

- The internal representation of Prolog terms and code has been redesigned, resulting in code that runs up to twice as fast as in release 3.
- Certain memory limitations that existed in release 3 have been dropped. All available virtual memory can be used without any limitations imposed by SICStus Prolog.
- The number of available atoms is four times larger than in release 3 (1M atoms are available on 32-bit platforms).
- The range of small integers is eight times larger than in release 3. Although the size of integers is unbounded, small integers are handled more efficiently than other numbers.
- Multifile predicates are compiled by default; in release 3, they could not be compiled.
- Native code compilation has been dropped.
- The profiling data accessible by `profile_data/4` and `library(gauge)` is more precise. Some of the choices of release 3 have been dropped.

8.1.2 Prolog Language

8.1.2.1 Single Language Mode

Release 3 had the notion of multiple language modes: `iso` and `sicstus`. Release 4 does not have this notion. The syntax and semantics of the Prolog language correspond to the previous `iso` language mode.

8.1.2.2 DCG Notation

The exact rules for translating DCG rules to plain Prolog clauses have not been laid down in a standard, but there is a broad consensus in the Prolog community about what they should mean. One of the guiding principles is that the translation should be steadfast, in particular that the translated code should always treat its last argument as an output argument and not use it “too early”. In some cases, a non-steadfast translation was produced in release 3. This has been corrected in release 4.

8.1.2.3 Asserting Terms with Attributed Variables

In release 3, terms containing attributed variables and blocked goals could be asserted, copied, and gathered as solutions to `findall/3` and similar predicates. The copy would contain new attributed variables with the attributes copied. This operation could be very expensive, could yield unexpected results and was not always safe e.g. in the context of CLPFD constraints. In release 4, the semantics of this operation has changed: in the copy, an attributed variable is simply replaced by a plain, brand new variable. Of course, if the same attributed variable occurs more than once, the same plain variable will occur in the corresponding places in the copy. If the attributes are relevant, the program can obtain them by using the new built-in predicate `copy_term/3` described below.

8.1.2.4 Arithmetic

The infix operator ‘#’ (bitwise exclusive or) has been renamed to ‘\’.

8.1.2.5 Syntax

Atoms can now contain the NUL character, i.e. character code zero. It is classified as white space and must therefore be entered using escapes. As an example ‘a\0a’ is a three character atom containing two a’s separated by a NUL.

Internally, atom names and other encoded strings, use the non-shortest form ‘0xC0 0x80’ to encode NUL. This is similar to how NUL is handled by Tcl/Tk and Java.

8.1.2.6 Prolog Flags

The `language` and `wcx` Prolog flag have been dropped.

The following Prolog flag is new:

`quoted_charset`

Controls the character set to use when writing quoted atoms.

8.1.2.7 Stream Properties

The `wcx` property has been dropped.

The following properties are new:

`encoding_signature/1`

Specifies whether an encoding signature (such as Unicode “byte order mark”) was used to determine the character encoding.

`encoding/1`

Subsumes the `wcx/1` option of release 3.

`eol/1`

Specifies how line endings in the file should be handled if the stream is opened in text mode.

8.1.2.8 Statistics Keywords

The following keywords are new:

`total_runtime`

Measures the total CPU time used while executing, including memory management such as garbage collection but excluding system calls.

`defragmentation`

Measures the number of and time spent performing memory defragmentation.

8.1.2.9 Built-In Predicates

The set of built-in predicates has changed slightly. The following predicates have been removed:

`'C'/3` This was used in the Prolog translation of DCG rules. It could trivially be replaced by unifications and served no other reasonable purpose.

`get0/[1,2]`
`put/[1,2]`

These used to have an overloaded semantics meaning one thing on binary streams and another thing on text streams. They have been subsumed by their ISO counterparts.

`get/[1,2]`
`tab/[1,2]`
`skip/[1,2]`

Although these do not have ISO counterparts, they have been removed for being in the spirit of `get0/[1,2]` and `put/[1,2]`. We have provided `skip_char/[1,2]`, `skip_code/[1,2]`, and `skip_byte/[1,2]` as an ISO style replacement for `skip/[1,2]`.

`ttyget0/1`
`ttyget/1`
`ttynl/0`
`ttyput/1`
`ttyskip/1`
`ttytab/1`
`ttyflush/0`

These used to exist as shorthands for the respective predicate with an additional `user` argument. In most cases, the “respective predicate” is one of the non-ISO style predicate mentioned above, so there was no point in keeping the shorthand.

`fileerrors/0`
`nofileerrors/0`

These used to exist as shorthands for `set_prolog_flag/2` with specific arguments, and so can be trivially replaced.

`call_residue/2`

Dropped because it was not possible to ensure the correct behavior in all circumstances, it relied heavily on copying terms with attributed variables, and it was not needed by any library module. It has been replaced by a similar predicate, `call_residue_vars/2`, which should suffice in most cases where `call_residue/2` was used; see below.

`undo/1`

Dropped because it was not possible to ensure the correct behavior in all circumstances. Users that know what they are doing can still call the unsupported predicate `prolog:undo/1`. The argument should have a module prefix.

`help/0`
`version/0`
`version/1`

These predicates, managing and displaying messages, can be easily emulated by features of the message system.

`fcompile/1`

`load/1` These predicates used to compile Prolog source code into ‘.ql’ files, and load such files. ‘.ql’ files serve a purpose when boot-strapping the Prolog system, but offer no advantages over ‘.po’ files, the Prolog object code format used by other built-in predicates.

`load_foreign_files/2`

This predicate provided a shorthand for building and loading a temporary foreign resource. Working with foreign resources is straightforward, and so the shorthand was dropped.

`require/1`

This predicate provided a shorthand for locating and loading library predicates. This was originally introduced for a compatibility reason that is now obsolete. It is straightforward to provide the necessary `:- use_module/2` directives, and so the shorthand was dropped.

The following predicates have been added:

`call/N` Generalizes `call/1`. For example, `call(p(1,2), a, b)` is equivalent to `call(p(1,2, a, b))`.

`skip_char/[1,2]``skip_code/[1,2]``skip_byte/[1,2]`

ISO style replacements for the non-ISO style `skip/[1,2]`.

`call_residue_vars/2`

Called as follows:

```
call_residue_vars(:Goal, -Vars)
```

Executes the procedure call *Goal*, unifying *Vars* with the list of residual variables that have blocked goals or attributes attached to them.

`copy_term/3`

Called as follows:

```
copy_term(+Term, -Copy, -Body)
```

Makes a copy of *Term* in which all variables have been replaced by new variables that occur nowhere outside the newly created term. If *Term* contains attributed variables, *Body* is unified with a term such that executing *Body* will reinstate equivalent attributes on the variables in *Copy*. Otherwise, *Body* is unified with `true`.

Some predicates have been changed slightly; in most cases, this affects predicates that take a list of options.

`is_mutable/1`

The predicate `is_mutable/1` has been renamed to `mutable/1`, in analogy with `integer/1`, `atom/1` etc.

module/1

The predicate `module/1` has been renamed to `set_module/1`, to avoid possible confusion with the `module/2` declaration.

format/[2,3]

For the predicate `format/[2,3]`, the semantics of the ‘~@’ spec has changed slightly: the goal `Arg` is called as if by `\+ \+ Arg`, i.e. any bindings made by the goal are lost.

close/2

Takes new options:

direction/1

Specifies which directions to close.

open/4

The `wcx/1` option has been dropped. Takes new options:

encoding_signature/1**encoding/1**

`eol/1` Correspond to the respective stream properties.

if_exists/1

Specifies what should happen if the file already exists.

absolute_file_name/3

The `ignore_underscores/1` option has been dropped. The `file_type/1` option value `q1` has been dropped. The `access/1` option values `execute`, `executable` and `search` are new. The `glob/1` option is new, allowing to match file names against a pattern.

load_files/2

The `load_type/1` option value `q1` has been dropped. `encoding_signature/1`, `encoding/1`, subsuming the `wcx/1` option of release 3, and `eol/1`, are new options, corresponding to the respective stream properties.

write_term/3

The `quoted_charset/1` option is new, reflecting the value of the Prolog flag with the same name.

halt/1

The predicate `halt/1` now raises an internal exception like `halt/0`. This gives surrounding Prolog and C code an opportunity to perform cleanup.

profile_data/4

The *Selection* argument now takes one of the values: `[calls,choice_points,instructions]`. The *Resolution* argument now takes one of the values: `[predicate,clause]`.

8.1.2.10 Hook Predicates

The hook `user:term_expansion/[2,4]` is replaced by the hook:

```
user:term_expansion(Term1, Layout1, Tokens,
                    Term2, Layout2, [Token|Tokens]).
```

The purpose of the new argument *Tokens* is to support multiple, independent expansion rules. The purpose of the arguments *Layout1* and *Layout2* is to support source-linked debugging of term-expanded code. Each expansion rule should have its unique identifying token *Token*.

The hook `user:goal_expansion/3` is replaced by the following per-module hook:

```
M:goal_expansion(Term1, Layout1,
                 Module, Term2, Layout2).
```

Typically, *Module* has imported the predicate *Term1* from module *M*. The purpose of the arguments *Layout1* and *Layout2* is to support source-linked debugging of goal-expanded code.

8.1.3 Library Modules

There is no consensus for a core library, portable across Prolog systems, let alone a standard for such a library. Since SICStus Prolog 3 was first released, SICStus has acquired Quintus Prolog, which has a rather rich library. For release 4, we have decided to make this asset be available to the SICStus community by providing a library that is a merger of the previous SICStus and Quintus libraries, which already overlap significantly.

The User's Manual documents the library of release 4. For the purposes of aiding code transition to release 4, the following is a list of the release 3 library modules, and their fate in release 4. See also [Section 8.2 \[Guide to Porting Code from Release 3\]](#), page 23.

atts	
comclient	
fdbg	
gauge	
heaps	
linda/client	
linda/server	
pillow	
prologbeans	
tcltk	
timeout	
trees	
wgraphs	
xml	As in release 3.
arrays	The native release 4 counterpart is called <code>library(logarr)</code> . Also available is a deprecated compatibility module <code>library(arrays3)</code> .
assoc	The native release 4 counterpart is called <code>library(avl)</code> , reflecting the abstract data type, AVL trees, and with a modified, richer API. Also available is a deprecated compatibility module <code>library(assoc3)</code> .

<code>bdb</code>	As in release 3, but uses the default Berkeley DB hash function, so all of the standard Berkeley DB utilities should now work.
<code>charsio</code>	Called <code>library(codesio)</code> in release 4. Likewise, the syllable ‘ <code>chars</code> ’ has been renamed to ‘ <code>codes</code> ’ in predicate names.
<code>clpq</code>	
<code>clpr</code>	As in release 3, unsupported.
<code>clpfd</code>	As in release 3, plus the following additions and changes:
<code>automaton/8</code>	is a new constraint capturing any constraint whose checker of ground instances can be expressed as a finite automaton.
<code>minimum/2</code>	
<code>maximum/2</code>	are new constraints, constraining a value to be the minimum (maximum) of a list of values.
<code>nvalue/2</code>	is a new constraint, constraining the number of distinct values taken by a list of values.
<code>cumulative/[1,2]</code>	provides a unified interface, subsuming <code>serialized/[2,3]</code> and <code>cumulative/[4,5]</code> .
<code>table/[2,3]</code>	defines an n-ary constraint by extension, subsuming <code>relation/3</code> .
<code>all_different/[1,2]</code>	
<code>all_distinct/[1,2]</code>	Arguments can have unbounded domains.
<code>scalar_product/[4,5]</code>	can optionally be told to maintain arc-consistency. This functionality subsumes <code>knapsack/3</code> .
<code>global_cardinality/[2,3]</code>	can optionally be told to use a simple algorithm. This functionality subsumes <code>count/4</code> .
<code>fd_copy_term/3</code>	is gone. Subsumed by built-in <code>copy_term/3</code> .
<code>jasper</code>	The <code>Jasper</code> module is not available in the current release. An alternative for Java users is <code>PrologBeans</code> .
<code>lists</code>	The native release 4 counterpart has a modified, richer API. Also available is a deprecated compatibility module <code>library(lists3)</code> .
<code>ordsets</code>	As in release 3, plus several new predicates.
<code>queues</code>	The native release 4 counterpart has a modified, richer API. Also available is a deprecated compatibility module <code>library(queues3)</code> .

- random** The native release 4 counterpart has a modified, richer API. Also available is a deprecated compatibility module `library(random3)`.
- sockets** The new predicate `socket_client_open/3` subsumes `socket/2` and `socket_connect/3`.
`socket_server_open/2` subsumes `socket/2`, `socket_bind/2` and `socket_listen/2`.
`socket_select/7` can wait for any kind of stream, not just socket streams. `socket_select/7` waits until one *unit* (character for text streams, byte for binary streams) can be transferred.
`socket_select/7` can wait for streams ready to write.
`socket_select/7` does not create streams, you need to explicitly use `socket_server_accept`.
Socket streams are binary by default.
Blocking socket operations can be interrupted on both UNIX and Windows.
`library(sockets)` should work with IPv6.
- system** Operations on files and directories have been moved to its own module, `library(file_systems)`. Process primitives have been redesigned and moved to a new module, `library(process)`. The predicates for creating temporary files, `mktemp/2` and `tmpnam/1`, have been removed. They used C library functionality that is broken by design and insecure. Instead, to create and open a temporary file use something like `open(temp('foo'), write, S, [if_exists(generate_unique_name)])`, possibly together with `stream_property(S, file_name(Path))` if you need to know the path to the generated file name.
The (little) remaining functionality is largely as in release 3. Also available is a deprecated compatibility module `library(system3)`.
- terms** As in release 3, plus several new predicates. `term_hash/2` is not guaranteed to compute the same hash values as in release 3.
- ugraphs** As in release 3, plus a couple of deletions.
- chr** A reimplementaion of `library(chr)`, based on the Leuven implementation.
- clpb**
- flinkage**
- objects**
- spaceout** Not present in release 4.
- vbsp** Not available in the current release. Visual Basic .NET and other .NET languages can use PrologBeans .NET.

The following is a list of library modules that are new in release 4.

- aggregate**
provides an aggregation operator for data-base-style queries.

<code>assoc</code>	uses unbalanced binary trees to implement “association lists”, i.e. extendible finite mappings from terms to terms.
<code>bags</code>	defines operations on bags, or multisets
<code>between</code>	provides some means of generating integers.
<code>file_systems</code>	accesses files and directories.
<code>objects</code>	provides a package for object-oriented programming, and can be regarded as a high-level alternative to <code>library(structs)</code> .
<code>process</code>	Process creation etc.
<code>rem</code>	provides Rem’s algorithm for maintaining equivalence classes.
<code>samsort</code>	provides generic sorting.
<code>sets</code>	defines operations on sets represented as lists with the elements unordered.
<code>structs</code>	provides access to C data structures, and can be regarded as a low-level alternative to <code>library(objects)</code> .
<code>types</code>	Provides type checking.
<code>varnumbers</code>	An inverse of <code>numbervars/3</code> .

8.1.4 Input-Output System

The internals of the I/O subsystem have been completely redesigned. The new version should be faster while at the same time providing more functionality and more consistent behavior between operating systems and between stream types.

The semantics of character codes has been fixed as (a superset of) UNICODE. Redefining the meaning of character codes is no longer supported.

New features and changes to the SICStus streams (`SP_stream`) include:

- Streams are binary or text also at the lowest level, e.g. in the C API, and there are separate operations for performing I/O of bytes and characters.
- Streams have a layered design. This makes it possible to add character set translation and other transformations (compression, encryption, automatic character set detection, ...) to any stream.
- All streams provide non-blocking operations and are interruptible, e.g. with `^C` (`'SIGINT'`). This is also true for file streams and under Windows.
- Subject to OS limitations, file names can use UNICODE and be of arbitrary length. In particular, under Windows, the UNICODE API is used for all operations.
- Limits on file size, file time stamps etc have been removed.
- Error handling has been simplified and made more consistent. In the C API all I/O operations return an error code from a rich set of error codes. Errors during write and close operations are no longer ignored.

- It is possible to wait for I/O ready (both for read and write) on any type of stream. This works for all platforms, including Windows. Select operations waits for the appropriate item type, e.g. until a whole (possibly multi-byte) character can be transferred on a text stream.

Other minor changes

- Now `byte_count/2` can be called only on binary streams.

8.1.5 Foreign Language APIs

8.1.5.1 Foreign Language Interface

The conversion specifier (in `foreign/[2,3]` facts) `string(N)` has been dropped.

The conversion specifier `chars` has been renamed to `codes`, in analogy with the built-in predicate `atom_codes/2`, the second argument of which is a list of character codes.

The C header generated by `splfr` from the `foreign/[2,3]` facts now uses the `const` attribute where appropriate.

Foreign resources are no longer unloaded by `save_program/[1,2]`. For this reason the `deinit` function of a foreign resource is no longer called when saving a program so `SP_WHEN_SAVE` has been removed.

8.1.5.2 C API Functions

Many functions in the C API has been changed or removed, especially those related to OS and I/O operations. There are also a number of new C API functions.

Old API	Replaced by
<code>SP_make_stream, SP_make_stream_context</code>	<code>SP_create_stream</code>
<code>SP_set_tty</code>	<code>SP_CREATE_STREAM_OPTION_INTERACTIVE</code>
<code>SP_fgetc</code>	<code>SP_get_byte, SP_get_code</code>
<code>SP_fputc</code>	<code>SP_put_byte, SP_put_code</code>
<code>SP_fputs</code>	<code>SP_put_codes, SP_put_encoded_string</code>
<code>SP_fflush</code>	<code>SP_flush_output</code>
<code>SP_chdir</code>	<code>SP_set_current_dir</code>
<code>SP_getcwd</code>	<code>SP_get_current_dir</code>
<code>SP_set_wcx_hooks</code>	Gone

<code>SP_wcx_getc, SP_wcx_putc</code>	Gone
<code>SP_to_os, SP_from_os</code>	Gone
<code>SP_put_number_chars</code>	<code>SP_put_number_codes</code>
<code>SP_get_number_chars</code>	<code>SP_get_number_codes</code>

Other new functions include

```
SP_get_stream_user_data
SP_get_stream_counts
SP_put_bytes
SP_fopen
SP_unget_code
SP_unget_byte
```

in addition many functions take new or changed parameters.

8.1.5.3 Java API

- The `Jasper` API has not changed from release 3, but the `Jasper` module is not available in the current release.
- The `PrologBeans` API has been extensively revised. See the `PrologBeans` HTML (javadoc) documentation.
`PrologBeans` was built with Java 1.5
-

8.2 Guide to Porting Code from Release 3

Release 4 does not provide a mode in which it is 100% compatible with earlier releases. However, this section provides guidelines for migrating Prolog code from release 3 to release 4.

1. First of all, make sure that your code runs in ISO execution mode. In release 3, the command line option `--iso` can be used.
2. A number of built-in predicates have been dropped. They are listed in the table below, along with their approximate substitutes. Refer to the documentation for each case.

Dropped built-in	Replaced by
<code>get0/[1,2], get/[1,2]</code>	<code>get_code/[1,2], get_byte/[1,2]</code>
<code>ttyget0/1, ttyget/1</code>	<code>get_code/2, get_byte/2</code>
<code>put/[1,2], tab/[1,2]</code>	<code>put_code/[1,2], put_byte/[1,2]</code>

ttyput/1, ttytab/1	put_code/2, put_byte/2
skip/[1,2]	skip_code/[1,2], skip_byte/[1,2]
ttyskip/1	skip_code/2, skip_byte/2
ttynl/0	nl/1
ttyflush/0	flush_output/1
fileerrors/0, nofileerrors/0	set_prolog_flag/2
'C'/3	unification
call_residue/2	call_residue_vars/2
undo/1	prolog:undo/1
help/0	the message system
version/0	the message system
version/1	the message system
fcompile/1	save_files/2
load/1	load_files/2
load_foreign_files/2	splfr + load_foreign_resource/1
require/1	use_module/2
is_mutable/1	mutable/1
module/1	set_module/1

3. The hook predicates `user:term_expansion/[2,4]` and `user:term_expansion/3` are now called `user:term_expansion/6` and `Module:term_expansion/5` and have a modified API; see [section “Term and Goal Expansion” in *the SICStus Prolog Manual*](#).

4. The set of library modules has been enriched by incorporating a subset of the Quintus Prolog library modules that we have deemed useful.

The following library modules are not included in SICStus 4: `jasper`, `clpb`, `flinkage`, `spaceout`. `library(objects)` has been replaced by its Quintus counterpart, with a completely different API.

The following table lists the affected SICStus 3 library modules.

Affected module	Closest equivalent	Comment
-----------------	--------------------	---------

<code>arrays</code>	<code>arrays3</code>	a
<code>assoc</code>	<code>assoc3</code>	b
<code>charsio</code>	<code>codesio</code>	c
<code>clpfd</code>	<code>clpfd</code>	d
<code>lists</code>	<code>lists3</code>	e
<code>queues</code>	<code>queues3</code>	f
<code>random</code>	<code>random3</code>	g
<code>sockets</code>	<code>sockets</code>	d
<code>system</code>	<code>system3</code>	h

Comments to the table:

- a. `library(arrays3)` is a code migration library module; the long-term solution is to use `library(logarrs)` instead.
- b. `library(assoc3)` is a code migration library module; the long-term solution is to use `library(av1)` instead.
- c. The syllable ‘chars’ has been changed to ‘codes’ throughout.
- d. Several API changes; see the documentation.
- e. `library(lists3)` is a code migration library module; the long-term solution is to use `library(lists)` instead.
- f. `library(queues3)` is a code migration library module; the long-term solution is to use `library(queues)` instead.
- g. `library(random3)` is a code migration library module; the long-term solution is to use `library(random)` instead.
- h. `library(system3)` is a code migration library module; the long-term solution is to use `library(system)`, `library(file_systems)` and `library(process)` instead.

8.3 Limitations in the Current Release

This section lists features that are missing or incompletely implemented in the current release of SICStus Prolog (SICStus Prolog 4.0.0) but that may appear in future releases. Please let us know what features are important to you!

No support for `SP_load_sicstus_run_time` and related features for loading multiple SICStus instances into the same process. This was mainly intended for the the, no longer supported, Jasper Java interface.

`library(tcltk)`: There is no way to pass non-Latin 1 characters from Tcl/Tk to Prolog. The Tcl/Tk Terminal is not supported.

`library(bdb)`: will not work reliably with non-ASCII file names.

`library(jasper)`: not supported but `library(prologbeans)` can be used as a partial replacement; see [Section 8.1.3 \[Library Modules\]](#), page 18.

`library(spaceout)`: not supported; see [Section 8.1.3 \[Library Modules\]](#), page 18.

The Visual Basic 6 module (`vbsp`) is not supported; see [Section 8.1.3 \[Library Modules\]](#), page 18.

The Windows “GUI” `spwin.exe` does not save or read any settings or command history. It also does not support full Unicode. The console version `sicstus.exe` fully supports Unicode when run from a console window.

The Emacs mode may not work reliably when passing Prolog code between Emacs and SICStus if the code is not written using Latin 1.

9 Generic Limitations

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

The number of atoms created may not exceed 1048575 (33554431) on 32-bit (64-bit) architectures.

The number of bytes making up the characters of an atom may not exceed 65535.

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

Saved-states are not portable between 32-bit and 64-bit architectures.

Indexing on large integers or floats is coarse.

10 Contact Information

Current support status for the various platforms as well as a web interface for reporting bugs can be found at the SICStus Prolog homepage:

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

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

The mailing list sicstus-users@sics.se is a mailing list for communication among users and implementors. To subscribe, write a message to majordomo@sics.se with the following line in the message body:

```
subscribe sicstus-users
```