SICStus Prolog Frequently Asked Questions

by the Intelligent Systems Laboratory

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

Release 3.10.1
April 2003

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

1 About This Document

This is a compilation of frequently asked question related to SICStus Prolog.

It is maintained by the SICStus Prolog support team. Any questions about this FAQ should be sent to sicstus-support@sics.se. Questions about SICStus Prolog should be addressed as follows:

- Bug reports and technical support questions to Report bugs through the web interface

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

  or to

  sicstus-support@sics.se
- Questions regarding licensing, contracts, etc. to sicstus-request@sics.se.

This FAQ is *not* intended for questions about the Prolog language in general. There is a specific Prolog FAQ for that, available at

http://www.cs.kuleuven.ac.be/~remko/prolog/faq/

2 Introduction

Question: What is SICStus Prolog?

Prolog is a simple but powerful programming language developed at the University of Marseille, as a practical tool for programming in logic. From a user's point of view the major attraction of the language is ease of programming. Clear, readable, concise programs can be written quickly with few errors.

SICStus Prolog is owned and maintained by the Swedish Institute of Computer Science. Parts of the system were developed by the project "Industrialization of SICStus Prolog" in collaboration with Ericsson Telecom AB, NobelTech Systems AB, Infologics AB and Televerket.

SICStus Prolog follows the mainstream Prolog tradition in terms of syntax and built-in predicates. As of release 3.8, SICStus Prolog provides two execution modes: the `iso` mode, which is fully compliant with the International Standard ISO/IEC 13211-1 (PROLOG: Part 1—General Core); and the `sicstus` mode, which is largely compatible with, e.g., C-Prolog and Quintus Prolog, supports code written in earlier versions of SICStus Prolog.

Question: Where can I get more information about SICStus Prolog?

The main source of information about SICStus Prolog is the WWW-site at

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

and in particular the User's Manual, Release Notes, and this FAQ list, at

`http://www.sics.se/isl/sicstuswww/site/documentation.html`

Question: Does SICStus run on platform XYZ?

SICStus runs under most UNIX dialects and under Windows 98/NT/2000. For a current list of supported platforms, see the release notes.

Question: What do I need to run SICStus?

- Disk space: 100 MB should be enough for packing up the binary installer and performing the installation. The actual installation will use less than half of that. 100 MB should also be enough to compile SICStus from the sources.
- Memory usage: At least 8-16 Mb RAM, but more memory is recommended.
- In order to use `library(tcltk)` you need a Tcl/Tk installation on your machine, see the release notes for information about what Tcl/Tk version you need and where you can get it.
- In order to use `library(jasper)` you need a JNI compatible Java installation. See the release notes for supported versions and where you can get it.

- In order to use `library(bdb)` you need a Berkeley DB. See the release notes for supported versions and where you gan get it.

Question: How can I get SICStus?

SICStus is a commercial product and licensing information can be found on the WWW-site (`http://www.sics.se/sicstus/`). Free evaluation licenses are available.

Upon completed license agreement, you will receive a mail containing download instructions including confidential passwords and encryption keys. Be sure to read the Release Notes before installation.

If you are a university student, your department may have acquired the right to sub-license SICStus to its students. If that is the case, your department should provide to you download instructions and license codes for the relevant distributions. Your professor should know.

Question: How do I report a bug or some other problem?

If you think you've discovered a bug or if you have another problem, first read this FAQ list, the release notes and the list of known bugs, at the Don't Panic page page at `http://www.sics.se/sicstus/`. If these do not address your problem, submit a bug report by filling in the form:

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

If you are using an academic license, for example if you are a student, then you should primarily consult with the maintenance contact at your site.

3 Installation problems

Some of the problems in this sections are UNIX (Windows) specific. These are marked *[UNIX]* (*[Windows]*).

3.1 Windows Installer Problems

Question: *[Windows NT 4]*

The installer complains that '`MSIEXEC`' cannot be found.

This happens if the installing user does not have '`Administrator`' rights. Either install as the user '`Administrator`' or ensure that the installing user have these rights during installation. If this is unclear, please contact your system administrator.

This problem should not happen on other versions of Windows.

3.2 Corrupted binary files

Question: *[UNIX]*

`gzip` says "stdin not in gzip format".

This means that the input piped to `gzip` has been corrupted in some way. The most likely cause is that the original file *`file`*`.tar.gz` was downloaded incorrectly, typically in ASCII mode instead of binary mode.

3.3 Linux C libraries

Question: *[UNIX]*

I get the message ".../sp.exe: no such file or directory" when running SICStus, even though `sp.exe` is there. What's wrong?

This error message appears when SICStus has been compiled using a different version of `libc.so` than your machine has installed. The message "no such file or directory" does not refer to `sp.exe` but to the particular `libc.so` version that SICStus expects.

SICStus is currently available as binary distributions for `libc.so.6`, a.k.a. `glibc`, versions 2.0, 2.1 and 2.2. Make sure to download the correct version.

3.4 Configure options

These issues are mainly of concern when building SICStus from a source distribution.

Question: *[UNIX]*
I've changed the arguments to `configure`, but nothing happens. Why?

The `configure` script maintains a cache-file called `config.cache`. This file has to be removed between configure-runs if you've changed the arguments. Alternatively, you can specify `--cache-file=/dev/null`, causing configure to avoid creating a cache-file entirely.

Question: *[UNIX]*
The configure script seems to ignore some of my options. Why?

Make sure that there is no site-wide configuration file (`config.site`). If configure finds one, it will print a message similar to:

```
loading site script /usr/local/etc/config.site
creating cache ./config.cache
checking SICStus version... 3.8.5
[...]
```

In this case, configure has found a `config.site` in `/usr/local/etc`.

The solution is to either remove the file or to set the environment variable `CONFIG_SITE` to an empty file of your choice. For example, assuming `csh`:

```
% setenv CONFIG_SITE ./config.site
% echo "# empty config.site" > ./config.site
```

3.5 Password problems

Question: *[Windows]*
When asked for the installation password, I typed it but it doesn't work. Why?

There are three pieces of information that you must not mix up.

- The password to the FTP/HTTP site containing the SICStus distributions.
- The password required by the installation program.
- Your license code, which comes with a site name and an expiration date.

3.6 Problems getting started

Question: I get the message "Failed to locate bootfile". What's wrong?

The message "Failed to locate bootfile" means that SICStus cannot find either `spds.sav` or `sprt.sav`, depending on whether you are executing a Development System or a Runtime System. If you are running a Runtime System, you may want to read the chapter "Runtime Systems" in the User's Manual. If you want to execute Runtime Systems on machines which

do not have SICStus installed (a.k.a. target machines), read the sections "Runtime Systems on Target Machines" in the Release Notes.

If you get this message running a Development System, your installation is probably inconsistent. Reinstall SICStus and try again.

Question: `^D` doesn't seem to bring me back a level down after `break`. What's wrong?

The manual assumes that `^D` is the EOF character. Under Windows, however, it's `^Z` (except when running under Emacs). In most contexts, the term `end_of_file` terminated by a full stop (`.`) can be typed instead of the EOF character.

4 Runtime problems

4.1 Memory allocation problems

Question: I get the message "Memory allocation failed"

A generic limitation of SICStus Prolog is that it cannot use more than 256 Mb of virtual memory on 32-bit architectures. This is an artifact of the tagged pointer scheme that we use. We have plans for replacing the tagging scheme, but it's a long term project. See the section "Generic limitations" in the Release Notes for additional details.

5 Generic Prolog Programming

Question: How can I split a module into several source code files?

You need a main file for the module, e.g. `main.pl` and several subfiles, e.g. `sub1.pl`, `sub2.pl`, ... Lay out the main file as follows:

```
:- module(ModuleName, ExportList).

... clauses/directives ...

:- ensure_loaded(sub1).

.. clauses/directives ...

:- ensure_loaded(sub2).
```

The subfiles can contain any clauses and directives, including `ensure_loaded/1` directives, but not `module/2` directives.

An alternative is to use an '`include declaration`'.

Question: How can I make the compiler abort the compilation if it encounters a syntax error?

This can be done by defining `user:message_hook/3` appropriately:

```
| ?- [user:user].
% consulting user...
| message_hook(error,_,Lines) :-
        print_message_lines(user_error,error,Lines), abort.
| end_of_file.
% consulted user in module user, 0 msec 24 bytes
```

This will intercept any error message, print the message, and abort:

```
| ?- [user].
% consulting user...
| p p.
! Syntax error
! operator expected after expression
! in line 39
! p
! <<here>>
! p .
% consulted user in module user, 0 msec -16 bytes
% Execution aborted
```

Question: How can I write access predicates for terms without paying a performance penalty?

There is support for unfolding predicates at compile time: `user:goal_expansion/3`. For example, assume that `is_ornode(X)` (`is_andnode(X)`) is true if the shape (principal functor) of X is `or/2` (`and/2`). If you consult the following:

```
:- multifile
        user:goal_expansion/3.

:- dynamic
        user:goal_expansion/3.

user:goal_expansion(is_ornode(Term), _, Term=or(_,_)).
user:goal_expansion(is_andnode(Term), _, Term=and(_,_)).

plan_tree( [N|_Rest], _GuidanceNodes, _Indent ) :-
        is_andnode(N).
plan_tree( [N|_Rest], _GuidanceNodes, _Indent ) :-
        is_ornode(N).
```

then `plan_tree/3` becomes transformed to:

```
plan_tree([A|_], _, _) :- A=and(_,_).
plan_tree([A|_], _, _) :- A=or(_,_).
```

If you are using `fcompile/1`, make sure that the definition of `user:goal_expansion/3`, and anything else that the compiler needs to know, has been loaded at fcompile time. A common idiom is:

```
?- ensure_loaded(SetOfFiles), fcompile(SetOfFiles).
```

Note that `fcompile/1` is obsolescent with the introduction of partial saved states (‘`.po`’ files).

Note also that `plan_tree/3` will not be able to determinately select a matching clause based on `A`, as predicates are indexed on the shape of the first argument only, which is a list in both clauses. Achieving indexing on `A` is the subject of the next question.

Question: Is SICStus capable of folding unifications across `:-` to determinately select a matching clause?

Consider the following clauses, with the above goal expansion:

```
plan_tree( [N|_Rest], _GuidanceNodes, _Indent ) :-
        is_andnode(N).
plan_tree( [N|_Rest], _GuidanceNodes, _Indent ) :-
        is_ornode(N).
```

In SICStus, as in most WAMs, indexing is done on the shape of the first argument. If all arguments are distinct variables *A*, *B*, *C*, ..., and the first goal is `A = Term`, indexing will be done on the shape of `Term`.

So to enable indexing on the shape of `N`, you must transform the clause e.g. to:

```
plan_tree( [N|_Rest], _GuidanceNodes, _Indent ) :-
        plan_tree_flat( N, _Rest, _GuidanceNodes, _Indent ).

plan_tree_flat(N, _Rest, _GuidanceNodes, _Indent ) :-
        is_andnode(N).
plan_tree_flat(N, _Rest, _GuidanceNodes, _Indent ) :-
        is_ornode(N).
```

With the above goal expansion, this code will indeed index on `N`.

Question: Why doesn't this Fibonacci program work?

Here's some code that purports to print all the Fibonacci numbers. The author of the code expected `retract/1` to backtrack forever, finding ever new `fibs/2` facts.

```
print_fibs :-
        retractall(fib(_)),
        assert(fibs(1,1)),
        retract(fibs(F1,F2)),
        write(F1), nl,
        F3 is F1+F2,
        assert(fibs(F2,F3)),
        fail.
```

If you run it:

```
| ?- print_fibs.
1

no
```

It doesn't work because of the semantics for calls to dynamic predicates in the presence of asserts and retracts. SICStus Prolog complies with the ISO Prolog standard in this respect. Clause 7.5.4 of the standard reads:

> Any change in the database that occurs as the result of executing a goal (for example, when the activator of a subgoal is a call of `assertz/1` or `retract/1`) shall affect only an activation whose execution begins afterwards. The change shall not affect any activation that is currently being executed.

In the above example, the `retract/1` goal is unaffected by the subsequent `assert`, and only succeeds once.

Question: The query `X=[97|X], name(A,X).` loops. Is it a bug?

It is possible, and sometimes useful, to write programs which unify a variable to a term in which that variable occurs, thus creating a cyclic term. The usual LP theory forbids the creation of cyclic terms, dictating that an *occurs-check* should be done each time a variable is unified with a term. Unfortunately, an occurs-check would be so expensive as to render Prolog impractical as a programming language. Thus cyclic terms may be created and may cause loops trying to print them.

SICStus Prolog mitigates the problem by its ability to unify, compare assert, and copy cyclic terms without looping. The `write_term/[2,3]` built-in predicate can optionally handle cyclic terms. Unification with occurs-check is available as a built-in predicate. Predicates testing (a)cyclicity are available in a library package. Other predicates usually do not handle cyclic terms well.

6 Interfacing with other Langauges

6.1 Interfacing with UNIX

Question: *[UNIX]*

How can I turn a Prolog source file into a UNIX script, to be invoked from the command line or from a Web application, for example?

Wrap the file into an executable Shell script which simply invokes SICStus. Informational messages can be filtered out by redirecting `stderr` to `/dev/null`. For example:

```
#!/bin/sh
exec sicstus 2> /dev/null -f -a "$@" <<EOF

% Tell SICStus to consult the sources.
% The embedded command invokes main(Args)
% where Args is the script's arguments passed as a list of atoms.

[user].

main(Args) :-
    format('Hello world!\n\c
            Invoked with args = ~q\n', [Args]).

:-
prolog_flag(argv, Args), main(Args).
EOF
```

The above script can be invoked as follows:

```
scheutz>./hw.sh a 236U ka=ka
Hello world!
Invoked with args = [a,'236U','ka=ka']
```

6.2 Interfacing with C++

See the Don't Panic page for some examples of how to interface SICStus with C++. The examples use Microsoft Visual C++, but the same technique can be adapted to other compilers (e.g. gcc on unix).

6.3 Interfacing with Visual Basic

See the manual and the release notes.

6.4 Interfacing with Tcl/Tk

6.4.1 Installing the Tcl/Tk library module under UNIX

NOTE: This section applies to UNIX binary distributions only.

As of 3.8, the Tcl/Tk library module is automatically configured by the installation script (`InstallSICStus`), after asking a few simple questions.

Question: When `InstallSICStus` asks for the "Tcl/Tk installation path", which path should I specify?

The installation path should be specified such that *installation-path*/lib contains the Tcl/Tk libraries (`libtcl<version>.so`). This is the same as the value of the `--prefix` options specified to Tcl/Tk's `configure`-script.

Question: My Tcl and Tk installation are not in the same directory. What should I do?

Unfortunately, there is no solution to this yet. You need to install Tcl/Tk in the same directory in order for `InstallSICStus` to find them.

6.4.2 Configuring the Tcl/Tk library module under Windows

Question: I need to use a different Tcl/Tk version from the one SICStus is compiled against. How do I do?

The symptom of this is that Windows complains about not finding `tcl<ver>.dll`.

See the release notes for what version of Tcl/Tk was used to build SICStus. If you want to use another Tcl/Tk version, you need to recompile `library(tcltk)`. You can do this by following these steps:

(Please do not embark on this unless you are fairly familiar with command-prompts and compilation procedures).

1. You need MSVC version 6.0 or later. 5.x or 4.x might work but have not been tested.
2. Build a new Tcl/Tk foreign resource (assuming Bash/Cygwin):

```
$ cd library/x86-win32-nt-4
$ mkdir tcltk_new
$ cd tcltk_new
$ splfr --cflag=-Ic:/path/to/Tcl/include ../../tcltk/*.c ../../tcltk.pl c:/path/to/
$ cp tcltk.dll ../tcltk.dll
```

   If you do not have a shell which expands "*.c", you need to replace "../../tcltk/*.c" with the name of all C source files in `../../tcltk`. Also, don't forget to adjust the actual paths to your Tcl/Tk installation directory.

The following sample session shows how it may look:

```
$ splfr --cflag=-Ih:/MS_Windows_2000/Tcl/include ../../tcltk/*.c ../../tcltk.pl h:/
SICStus 3.10.0 (x86-win32-nt-4): Sat Jan 11 15:04:03  2003
Licensed to sics

% tcltk_glue_1084_1044046414.c generated, 20 msec
% tcltk_glue.h generated, 20 msec
tcl.c
../../tcltk/tcl.c(223) : warning C4090: 'function' : different 'const' qualifiers
../../tcltk/tcl.c(223) : warning C4024: 'sptcl_save_error' : different types for fo
../../tcltk/tcl.c(323) : warning C4090: 'function' : different 'const' qualifiers
../../tcltk/tcl.c(323) : warning C4024: 'sptcl_save_error' : different types for fo
../../tcltk/tcl.c(374) : warning C4090: 'function' : different 'const' qualifiers
../../tcltk/tcl.c(374) : warning C4024: 'sptcl_save_error' : different types for fo
tk.c
../../tcltk/tk.c(134) : warning C4090: 'function' : different 'const' qualifiers
../../tcltk/tk.c(134) : warning C4024: 'sptcl_save_error' : different types for for
../../tcltk/tk.c(162) : warning C4090: 'function' : different 'const' qualifiers
../../tcltk/tk.c(162) : warning C4024: 'sptcl_save_error' : different types for for
tkappini.c
tkterm.c
util.c
tcltk_glue_1084_1044046414.c
   Creating library dummy.lib and object dummy.exp
$ cp tcltk.dll ../tcltk.dll
```

3. Test your new resource:

```
$ sicstus -i
SICStus 3.10.0 (x86-win32-nt-4): Sat Jan 11 15:04:03  2003
Licensed to sics
| ?- use_module(library(tcltk)).
% loading c:/program files/sicstus prolog 3.10.0/library/tcltk.po...
% module tcltk imported into user
%  loading foreign resource c:/program files/sicstus prolog 3.10.0/library/x86-win3
% loaded c:/program files/sicstus prolog 3.10.0/library/tcltk.po in module tcltk, 1
yes
```

6.4.3 Compiling SICStus with Tcl/Tk support

NOTE: This section is only relevant when compiling source distributions with support for Tcl/Tk.

Most UNIX machines today have Tcl/Tk preinstalled. This reduces the risk of installation difficulties since the linker and the runtime linker usually finds Tcl/Tk in its default paths. Most other installation and/or configuration problems related to Tcl/Tk are caused by

improper installation or installation in a path where the linker and runtime linker cannot find Tcl/Tk.

Tcl/Tk is configured for installation similar to SICStus; by the use of a `configure` script generated by GNU Autoconf. The installation directory is specified by the `--prefix` option. The value of this option is referred to as the *installation-dir* below.

Question: The configure-script complains that it cannot find `Tcl_Init` and/or `Tk_Init`. What should I do?

The following message from `configure` indicates that it could not link a test program with the Tcl/Tk libraries.

```
...
checking for Tcl_Init in -ltcl8.3... no
checking for Tk_Init in -ltk8.3... no
...
```

The common cause of this is that the compiler cannot find `-ltcl8.3` and/or `-ltk8.3` (or whichever version it tries to find), usually because Tcl/Tk has been installed in a non-standard directory (i.e. different from `/usr/lib`, `/usr/local/lib`, etc.).

If you've installed Tcl/Tk in *installation-dir*, specify the following to `configure`:

```
% ./configure --with-tcltk=installation-dir more options
```

The problem can occur even if the Tcl/Tk shared libraries can be found, but the link-phase failed due to other reasons. If this occurs, it is often useful to look in `config.log`, where output from compilation stages etc. is logged.

Question: I've installed Tcl and Tk in different directories. What do I do?

You can use:

```
% ./configure --with-tcl=tclinstallpath --with-tk=tkinstallpath
```

Question: There are no shared Tcl/Tk libraries. What has happened?

You need to specify `--enable-shared` when configuring Tcl/Tk. This is *not* enabled by default.

6.4.4 Generic Tcl/Tk problems

Question: I get the message "tcl_new/1 - Can't find a usable init.tcl in the following directories..."

This means that Tcl/Tk can't find its libraries of tcl-code needed for operation. This occurs when Tcl/Tk has been built but not properly installed. You have the option of

stating the location of the libraries by means of the environment variables `TCL_LIBRARY` and `TK_LIBRARY`. See also the Tcl/Tk documentation.

Another reason for this could be if you are not using SICStus 3.8.1 or later. The original SICStus 3.8 did not initialize Tcl/Tk correctly.

Question: I get the message "tcl_eval/3 - can't read 'var(Tree)': no such variable"

This message can appear in Tcl code which tries to access Prolog variables. Example:

```
tk_test :-
    tk_new([],Interp),
    tcl_eval(Interp,
             '
              global var
              set Tree []
              set Children []
              set GrandChildren []
              prolog {get_tree(Tree),
                     make_children(Tree, Children),
                     make_grandChildren(Tree, Children, GrandChildren)}
              puts "$var(Tree) $var(Children) $var(GrandChildren) "
              ', _),
    tk_main_loop,
    tcl_delete(Interp).
```

The problem is that the argument to the `prolog` function is just a string that is interpreted by Prolog as a goal. There is no connection between the Prolog variables `Tree`, `Children` and `GrandChildren` and the Tcl-variables with the same name. After returning from `prolog`, the values which the Prolog variables have been bound to are assigned to the array variable `prolog_variables`.

So replacing `$var` in your example with `$prolog_variables`, makes it work.

6.5 Interfacing with Java

Question: I'm having basic trouble in getting Java to work. What should I do?

First try to get a simple "Hello world" program to work that does *not* use Jasper. A large portion of the support questions we get are unrelated to the SICStus Java interface. Once you can compile and run your own Java classes it should be relatively easy to get Jasper to work.

Read the Release Notes and the User's Manual and the section in them related to Jasper.

Question: I want to use Jasper from an applet, but my browser won't let me. What's wrong?

Jasper is built as a layer on top of SICStus' C-Prolog interface and hence Jasper needs to be able to call methods declared as *native* (i.e., C functions). For security reasons, this is prohibited in applets.

This is a serious restriction, and future versions of Jasper will most likely allow connecting to the Prolog engine via sockets. In the interim, there are some examples on this at the Don't Panic page at `http://www.sics.se/sicstus/`.

Question: I get the message "Failed to locate bootfile" when running `java` *`ClassUsing-Jasper`*. What's wrong?

The answer to this (and related questions) can be found in the Jasper chapter of the Release Notes, section "Getting Started".

7 Standard Prolog compliance

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

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

8 Miscellany

Question: *[Windows]*

How do I save a transcript of my Prolog session on Windows?

As of 3.8.3 the Windows console (‘`spwin.exe`’) can 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).

Question: *[Windows]*

How do increase the number of history lines used by the Windows console (‘`spwin.exe`’)?

See section “The console” in *the SICStus Prolog Release Notes*.

Table of Contents