CCLRC / Rutherford Appleton Laboratory
Particle Physics & Astronomy Research Council
Starlink Project
Starlink User Note 248

**SUN/248.0**

Norman Gray
Steve Rankin
Peter Draper
29 April 2005

---

# Building Starlink Software

## Abstract

This document contains instructions on how to build the Starlink software from source. It contains a few platform-specific notes, and a comparison with the old 'mk-based' build system. This document is deliberately terse, since we have invested effort in making the processes involved as easy as possible.

This is at present a draft document, and so you should expect it to change relatively frequently.

$Revision: 1.1 $
$Date: 2005/04/29 11:44:30 $

*Document information*
Document date 29 April 2005
Version number 0

# Contents

# 1 Introduction

Starlink distributes its software products in a variety of ways.

The project makes periodic binary releases. These are made every six months in principle, though the intervals are usually somewhat longer in fact. The set of platforms we target is not fixed, but in the latest release the list was Linux RHEL3, RH9 and Debian3.0, and Sparc 8/9, with a beta release for Mac OS X. For fuller details, see the 'Software' link on the Starlink home page[1].

All the software in such a release is also available as source tarballs, and some components of the software set make interim source releases, also as source tarballs. This document is about how to build, and once built how to link against, these distributed source tarballs. It is a short document, since it is our intention that building and installing our software should be as simple as

```
% ./configure; make; make install
```

exactly as for the majority of open-source software available now. Execpt where some technicality prevents it, all the Starlink libraries are installed as both static and shared libraries, and there should be no surprises in linking agsinst them. The only difference from the standard behaviour is that the default installation prefix for our software is in either `/star` or `/stardev` (the latter for interim releases or beta software), rather than `/usr/local`. You can confirm this with `./configure --help`.

The software should build without problems, as we have invested considerable effort in making it portable across platforms and compilers. As with any other very large software system, however, we are aware of some wrinkles, and these are discussed in the platform-specific notes in *Section 2* below. If you find any other problems, or have other comments on the build system, please do get in touch with the project, through the general contact address `ussc@star.rl.ac.uk`[2].

You may be used to Starlink's earlier build system, based on `./mk` scripts. This is now obsolete, and a migration path is outlined in *Section 3* below

## 1.1 The Starlink CVS repository

All of Starlink's source code is kept in a CVS repository, which is publicly available. The Starlink build system consists of

- the Starlink CVS repository;
- build tools, consisting of modified versions of the GNU autotools `autoconf` `automake` and `libtool`;
- configuration and bootstrap code, consisting of the `starconf` system and the scripts at the top level of the Starlink source tree.

[1] `http://www.starlink.ac.uk`
[2] `mailto:ussc@star.rl.ac.uk`

The repository is on the machine `cvs.starlink.ac.uk`. See `<http://www.starlink.ac.uk/Download/getcvs.html>` for details of checking out the software and building the system.

The `getcvs.html` web page above says that, after checking the source tree out of CVS, you should read the `README` at the top of the tree. There are platform-specific notes about this in *Section 2* below.

The build system as a whole is described in considerable detail in SSN/78. This is a large document, but you probably don't have to read it, if what you want to do with the Starlink applications and libraries is to build them from distributed source tarballs, use the applications, and link against the libraries from your own programs. You will only need to read SSN/78 if you wish to

1. build from the up-to-the-minute versions in the CVS repository (though even in that case, you are probably better with the up-to-the-day versions available in the nightly build tarballs);
2. work on the CVS sources themselves, since you are welcome to play with the versions of the sources available as an anonymous CVS checkout; or
3. you wish to add further components to the Starlink CVS repository, though in this case you should probably talk to the project first, to coordinate your efforts with others.

# 2 Platform-specific notes

To build the software set, you should follow the `README` at the top level of the checked-out tree. You can examine the current version of this file, and its history, at `<http://cvsweb.starlink.ac.uk/cvsweb.cgi/README>`.

## 2.1 Building on OS X

You need a Fortran compiler: although the system compiler on OS X is GCC, Apple don't include g77 in their distribution.

You can get a g77 package from either DarwinPorts or Fink. Darwinports is (in Norman's experience) rather more fiddly to set up than Fink, but it is possibly to be preferred, since it has a certain amount of Apple support and contributions from Apple employees; it appears to be more deliberately aligned with the GCC version in the OS X distribution; and there are suggestions that Apple may at some point 'bless' it, and make it more easily integrated with a standard OS X setup.

Once you've installed one of the g77 packages, and put the appropriate binary directory in your PATH, you should be ready to go.

If you want to build and install the documentation then you should install TeX on Mac OS X (get the `gwTeX` distribution (good) from `<http://www.rna.nl/tex.html>`; there are other notes about TeX on OS X at `<http://www.esm.psu.edu/mac-tex/>`). Otherwise use:

```
% ./configure -C --disable-shared --without-stardocs
```

You might also have to install ghostscript from Darwinports or Fink.

You should also install the Apple X11 distribution and also the X11-SDK, this should be included on your Mac OS X distribution media, but not installed by default. Do not use the Darwinports or Fink X11.

If you want to build STARJAVA, then you will need to install the Java Advanced imaging and 3D support from Apple.

## 2.2 Building on Tru64

The configure script will choose a Fortran 95 compiler before it settles for a Fortran 77 one. Some of the code requires a Fortran 77 compiler (at least at present, possibly indefinitely), and so if you have more than one Fortran compiler installed, you'll have to force the build system with a command like:

```
% ./configure FC=f77 [other arguments]
```

where the Tru64 Fortran 77 compiler is called `f77`. You can give the path to the compiler also, if the f77 you want isn't the first one in your `PATH`.

The Tru64 C and C++ compilers should work fine: if you have more than one compiler installed, you can force them with the configure arguments `CC=cc` and `CXX=cxx`, but this shouldn't be necessary.

### 2.3 Building on Solaris

The configure script will choose a Fortran 95 compiler before it settles for a Fortran 77 one. Some of the code requires a Fortran 77 compiler, which means that, if you have more than one Fortran compiler installed, you'll have to force the build system with a command like:

```
% ./configure FC=f77 F77=f77 FPPFLAGS='-P -fixed' [other arguments]
```

where the Solaris Fortran 77 compiler is called `f77`,. You can give the path to the compiler also, if the f77 you want isn't the first one in your `PATH`.

This dependence on a Fortran 77 compiler is true at present, and may be the case indefinitely; at least part of the problem is libtool, which appears not to understand f95 on Solaris, so this problem might go away with a libtool update. As libtool doesn't understand the `FC` flag, the `F77` variable has to be set to match the `FC` one.

The Solaris C and C++ compilers should work fine: if you have more than one compiler installed, you can force the choice with the configure arguments `CC=cc` and `CXX=CC`, but this shouldn't be necessary. The system works with gcc and g++ on Solaris, too, and these compilers can be selected using the same mechanism.

### 2.4 Building on Cygwin

The configure script fails to identify how to preprocess Fortran, so it necessary to set environment variables with a command like:

```
% ./configure FPP=g77 FPPFLAGS='-E' [other arguments]
```

### 2.5 Building on x86

There are no known problems building on x86 boxes.

Whoopee, and all that.

### 2.6 Building on 64-bit x86

The only known problem on x86_64 is a distribution-specific one (I think). On RHEL, the tetex 1.* package installs the kpathsea library as a static library alone, and not a shared one. This produces an error message when compiling the `dvi2bitmap` component, resembling:

```
ld: ./libkpathsea.a(tex-file.o): relocation R_X86_64_32 can not be used when making
a shared object; recompile with -fPIC
./libkpathsea.a: could not read symbols: Bad value
```

To address this, you can either configure `dvi2bitmap` using `--without-kpathsea` at some loss in functionality, or else recompile the kpathsea library yourself. The sources are available at TUG[3] -- you need to rebuild only the kpathsea part of this distribution: to do that:

[3] `http://www.tug.org/ftp/tex/web2c.tar.gz`

1. Unpack the distribution, and go to the `web2c-x.x.x` directory
2. Configure using `./configure --enable-shared --datadir=XXX --prefix=PREFIX`, where `XXX` is the location of teTeX on your local machine (probably `/usr/share` on RHEL) and `PREFIX` is where you would like the newly-built library to be installed.
3. cd to the kpathsea directory, and delete the lines in the Makefile that refer to (unwritable) `$(web2cdir)`
4. `make; make install` to install in `PREFIX`.

The installation directory `PREFIX` will have to be in your `LD_LIBRARY_PATH` when you re-configure `dvi2bitmap`.

# 3 Migration from a ./mk based build system

The Starlink code set is now built using the standard (or at least, conventional) autoconf system, so that you configure the system before building using the command `./configure`. See the sections above for more discussion.

This is rather different from the old Starlink system, which used a mk script, which set up a number of environment variables to platform-specific values and then invoked make on your behalf. The `mk` script consisted of a switch which set appropriate values for each of the platforms which the project supported (plus sometimes a few other contributed ones). The configure-based system should be more portable, and work on a broader range of platforms without such platform-specific customisation.

If you have previously built software using the mk-based system, you will see little difference in the new system, beyond the difference in procedure described above.

If, on the other hand, you have previously developed using the mk-based system, there will be a few more changes:

1. mk scripts will no longer work
2. The `INSTALL` and `SYSTEM` environment variables have disappeared.
3. The `xxx_dev` scripts have disappeared.
4. The system now uses shared libraries extensively, and you might have to indicate where these are.

If you are simply linking against Starlink libraries, you probably don't have to care about any of this, and the only difference is that you now have shared libraries and centrally located include files to make your job easier. Most of the libraries in the distribution are built using `libtool`, and the libtool `.la` 'library files' are installed.

If you are planning to work on applications in the CVS repository, you should probably take a look at SUN/78, and might want to introduce yourself on the Starlink developers mailing list, at `<http://www.jiscmail.ac.uk/lists/stardev.html>`.

## 3.1 mk scripts

mk scripts are now obsolete. If you have made or modified a `mk` script of your own, you might possibly need to rework things to fit in with the new system. SUN/78 describes the new build system in detail, along with introductions to the autotools and pointers to fuller documentation.

## 3.2 The INSTALL and SYSTEM variables

The `INSTALL` and `SYSTEM` variables have disappeared.

The `SYSTEM` variable used to specify the system on which the software was being built, and the `mk` script switched on this value to make its settings, making the assumption that the platform reliably implied the compilers being used. This is now redundant, since the autoconf-based system makes its decision, not on the basis of what platform it discovers it is running on, but

on the basis of the tools and capabilities it dynamically discovers. This variable is now simply ignored.

The `INSTALL` variable used to specify the installation location. This role is now performed by the `--prefix...=` option to the `./configure` script. This script will, however, look at the `INSTALL` variable and, if it is set, expect it to name a program to use instead of the standard `/usr/bin/install` program, to install the software after building. You should almost certainly *not* have this variable defined, and if it is defined by accident, to anything other than an installation program, the buildsystem will probably fail messily.

See the top level `README` file for further details about environment variables which affect the build.

### 3.3 The xxx_dev scripts, and include files

The `xxx_dev` scripts and include files have changed.

In the `mk` system, the `xxx_dev` scripts had to be used to make soft links to the (Fortran) include files necessary for working with a library. In the new system, all such include files are installed in a standard place, such as /star/include. All the Fortran compilers that we are aware of have a `-I` option to indicate where such include files are to be found.

Note that the Fortran include files are now consistently upper-case.

### 3.4 Shared libraries

With only one or two exceptions, all the Starlink libraries are built and installed as shared libraries, as well as static ones. You might, therefore, have to do something special to link against these. On many Unixes, the environment variable `LD_LIBRARY_PATH` specifies a list of directories which are searched for shared libraries, in addition to a set of system defaults. The analogue on Mac OS X is `DYLD_LIBRARY_PATH`.

The build system uses libtool[4] extensively, and it installs its 'libtool library files', `wibble.la`, in the library directory alongside the more usual static `.a` and dynamic `.so` files (as appropriate for your platform). If you also use libtool, you can usefully link against these `.la` files yourself.

See your system's `ld` man page for more details than you can shake a stick at.

## Change history

### Version 0

Norman Gray, 29 April 2005

### Changes in version 0

Norman Gray, 29 April 2005

Draft release: $Revision: 1.1 $

[4] `http://directory.fsf.org/libtool.html`