

19. SHELX-97 Installation

Before trying to install the programs, it is worth checking with the SHELX homepage at <http://linux.uni-ac.gwdg.de/SHELX/> to see if there are any last-minute changes and whether other users have encountered problems on particular machines. The ftp site and CDROM contain the following files and subdirectories:

readme - Installation instructions, last-minute changes, changes since SHELXL-93 etc..

shelx.htm and **shelxman.htm** - On-line help in HTML format: requires a browser such as Netscape. **shelx.htm** contains the same general information as in README, and calls **shelxman.htm** that includes summaries of commands etc. **applfrm.htm** is the application form in html format. The file extensions will need to be changed to .html to active these files. three-letter extensions are used in the release for compatibility with MSDOS.

Subdirectory **'unix'** contains the sources of all programs for relatively standard UNIX systems. These should also compile successfully on many other operating systems too (except VMS).

Subdirectory **'vms'** contains the VMS sources for Digital computers.

Subdirectory **'doc'** contains the full manual in WINWORD 6 format, one file per chapter. It is designed to print on letter sized paper.

Subdirectory **'ps'** contains the full manual in Postscript format, one file per chapter. It is designed to print on letter sized paper.

Subdirectory **'egs'** contains the test jobs and other examples files.

Subdirectory **'ibm'** contains the IBM RS6000 executables (these also execute on the IBM Power-PC series).

Subdirectory **'sgi'** contains the SGI IRIX executables; they should run under IRIX 5.3 or later with the R4000 series processors. For other systems it is desirable to recompile to obtain programs that execute faster even if the precompiled versions run correctly.

Subdirectory **'linux'** contains the LINUX executables for Intel processors.

Subdirectory **'dos'** contains the pure MSDOS executables. These may or may not run in the DOS windows under WINDOWS or OS2.

In addition, the ftp login directory contains gzipped tar files of the above subdirectories (e.g. **unix.tgz**). These are convenient for down-loading with ftp as shown in the next section. The current sizes of these files in bytes are given on the SHELX homepage and should be checked to ensure that transmission is complete.

19.1 Installing the precompiled versions

In many cases it will be possible to use the precompiled versions provided. The executable programs (and the file `ciftab.def`) should simply be copied from the appropriate directory on the CDROM or ftp site to a directory on your machine. This directory should be specified in the 'PATH' so that the executables can be found. On UNIX systems the lazy way is to copy the programs into `/usr/bin`; on MSDOS systems they are usually copied to `C:\EXE` and this directory name is then added to the PATH specified in `AUTOEXEC.BAT`. You may also wish to copy the documentation and examples files.

As an example we shall take a PC running Linux; the following files should be fetched to your working directory by ftp (binary transfer !); for most other UNIX systems the installation procedure is similar:

linux.tgz, *ps.tgz*, *egs.tgz*, *shelx.htm* and *shelxman.htm*

The three gzipped tar files can then be expanded:

```
gunzip *.tgz
tar -xvf ps.tar
tar -xvf egs.tar
tar -xvf linux.tar
```

which will create the subdirectories *ps*, *egs* and *linux*. The executables can be copied to `/usr/bin` (needs system manager priviledges !):

```
cp linux/* /usr/bin
```

Under LINUX it is particularly easy to print the documentation, because `lpr` can recognize and print Postscript even on a non-Postscript printer:

```
lpr ps/*.ps
```

The on-line help files *shelxl.htm* and *shelxman.htm* should be renamed (`mv`) to *shelxl.html* and *shelxman.html* (the three-letter extension was needed for MSDOS systems !) and copied to a generally accessible directory; they may then be viewed with Netscape or any other HTML browser. These files are NOT copyrighted and you are welcome to improve and extend them as you wish for non-commercial purposes. *shelx.htm* calls *shelxman.htm* and *applfrm.htm* (the application form) It contains all the information from 'README' (which is a plain ASCII text file) plus a summary of the documentation (the full documentation is available in WINWORD 6 format in subdirectory '*doc*' and in Postscript form in subdirectory '*ps*').

19.2 Program compilation under UNIX (and other operating systems)

The UNIX version has been designed to be easy to compile on a wide range of UNIX (and other) systems. The resulting compiled programs do not need any environment variables or hidden files to run; it is simply necessary that the executable program is accessible via the PATH or an alias. The simplest way is to copy the executables into `/usr/bin`.

The SGI executable of SHELXL was compiled under IRIX 5.3 as follows:

```
f77 shelxl.f -O3 -c
f77 shelxlv.f -O3 -c
f77 shelxl.o shelxlv.o -o shelxl
```

The compilation for other UNIX systems should be similar. **IT IS NECESSARY TO BE VERY CAREFUL ABOUT OPTIMIZATION.** The safest is to compile without any optimization first (-O0 rather than -O3 in this case), run the ags4 and 6rxn tests, and rename the resulting output files **.res*, **.lst*, **.fcf* and **.pdb*. Then recompile with highest optimization (-O3), rerun the tests, and use the UNIX diff instruction to compare the results with those from the unoptimized version. Small differences in the last decimal place do not matter, and of course the CPU times will differ, but if there are significant differences then the optimization level should be lowered and the tests repeated. For some systems (including certain SG Challenge and Digital Alpha systems), only the shelxlv.f file (containing the rate-determining routines) can be compiled with the highest optimization level; shelxl.f must be compiled at a lower level.

The *shelxl.f* source contains the following routines that may be different or not available for some FORTRAN compilers:

IARGC and GETARG: these have always worked so far; if necessary the standard C routines could be adapted since the specifications are the same.

EXIT and FLUSH: if these cause problems they can safely be commented out in the source or replaced by the dummy FORTRAN subroutines provided. EXIT is used in 2 places to tidy up before terminating, FLUSH(6) occurs once to flush the logfile so that a batch job can be watched as it runs (eg. with tail -f).

ETIME and FDATE: most UNIX FORTRAN compilers will recognize these routines. For compilers that do not, both FORTRAN and C substitutes are provided. Usually at least one substitute will work, but the following points should be checked carefully:

Some FORTRAN compilers add an underscore to the end of procedure names before searching them in a library (this avoids confusion with standard C routines that happen to have the same names). The C versions are provided both with underscores (files *fdate_.c* and *etime_.c*) and without (*fdate.c* and *etime.c*).

The FORTRAN substitute for FDATE (*fdate.f*) calls FORTRAN routines TIME and DATE. Some compilers link in the C procedure 'time' instead, with strange results because the parameters may be different. The alternative *fdate.c* is safer.

The C replacement for ETIME (*etime.c*) may suffer from time 'wrap-around' if a large value for CLOCKS_PER_SEC (say 1000000) is combined with the use of a 32-bit or shorter integer to pass the time (!). Check the type time_t and CLOCKS_PER_SEC in */usr/lib/sys/time.h* (you may need to consult a Guru).

The IBM RS6000 executable was compiled as follows; note that *fdate.f* cannot be used for the reason given above, and that the underscore is not needed after *fdate* in the C subroutine. The FLUSH routine was replaced by the dummy.

```
xlf shelxl.f -O -c
xlf shelxlv.f -O -c
xlf etime.f -c
xlf flush.f -c
xlc fdate.c -c
xlf shelxl.o shelxlv.o fdate.o etime.o flush.o -o shelxl
```

The Linux executable was compiled using the GNU FORTRAN and C compilers as follows. The Absoft compiler is not recommended because optimization gives bad numerical results, and f2c produces slower code. The two C routines require underscores.

```
g77 shelxl.f shelxlv.f etime.c fdate.c -O3 -ffast-math -o shelxl
```

SHELXS uses the same routines and should be compiled just like SHELXL. The rate-determining routines are in *shelxsv.f*, the rest in *shelxs.f*.

One commented line near the start of SHELXL and SHELXS needs to be changed if these programs should write MSDOS format ASCII text files rather than UNIX format when run on a UNIX system. This is useful for a heterogeneous UNIX/MSDOS network, because the UNIX versions of all SHELX programs can read MSDOS format files. but not vice versa.

The remaining programs do not require optimization (except possibly SHELXA and SHELXPRO) and do not require FDATE, ETIME, FLUSH and EXIT, so they are easier to compile. For example under IRIX 5.3:

```
f77 shelxpro.f -O3 -o shelxpro
f77 shelxwat.f -o shelxwat
f77 ciftab.f -o ciftab
f77 shelxa.f -O3 -o shelxa
```

Unlike SHELXL and SHELXS, there are some intentional deviations from the strict FORTRAN-77 standard in these programs. REAL*8 and list-directed reading of internal files are used in several cases, and SHELXPRO uses types INTERGER*2 and BYTE in order to produce binary map files for O. Most FORTRAN compilers have no problems with these extensions, but may output warning messages.

Note that CIFTAB will search the current directory for a specified format file, and if it doesn't find it there it will look for it in a directory that is defined in the source. Unless this is edited before compiling, the directory is set to /usr/bin, so if the executable programs are located in /usr/bin the file ciftab.def (the default format file) should be there too.

19.3 Program compilation under VMS

The following instructions may be tried for compilation of the VMS sources under OpenVMS:

```

$fort/opt/ass=(noac,nodu)/align=all shelxs+shelxsv
$link shelxs
$fort/opt/ass=(noac,nodu)/align=all shelxl+shelxlv
$link shelxl
$fort/noopt shelxpro
$link shelxpro
$fort/noopt shelxwat
$link shelxwat
$fort/noopt shelxa
$link shelxa
$fort/noopt ciftab
$link ciftab

```

It may be necessary to split up the programs into subroutines to prevent the compiler running out of virtual memory. The files produced by the test jobs for SHELXL and SHELXS MUST be compared with those obtained using unoptimized versions of these programs (compiled with /noopt instead of /opt; note that /opt is usually the default) since optimizing errors are common for Digital compilers; there is a DIFF instruction in VMS that can be used for this. The remaining programs are not very CPU-intensive and so should not be optimized. If optimization causes errors, it is worth trying just to optimize *shelxsv.f* and *shelxlv.f* (which contain the rate determining routines) but not the rest. The executables need to be defined as follows:

```

shelxs ::= $ disk:[directory]shelxs   etc.

```

where 'disk' and 'directory' should be replaced by the appropriate local names and the programs are run (after preparing the files name.ins and name.hkl) by e.g.

shelxl name

SHELXWAT and SHELXA accept UNIX-type switches (even under VMS); they MUST come before the filename, e.g.

shelxwat -h name

No other files or parameter settings are required to run the programs, except that the file ciftab.def or a user-produced format definition file should be in the current directory when CIFTAB is run; if this file cannot be found in the current directory, CIFTAB searches for it in a directory specified in the source.

19.4 Parallel and vector machines

SHELXL and SHELXS are designed to run very efficiently on vector computers (such as older Cray and Convex machines); no changes should be needed to the code. Unfortunately the crystallographic algorithms involved are less suitable for parallel computers (or multiprocessor systems); in such cases the available computer resources are more efficiently used by running several jobs simultaneously, one per processor.

19.5 SHELXH - version of SHELXL for very large structures

SHELXH is a special version of SHELXL for the refinement of very large structures (with more than about 10000 unique atoms). The only difference between shelxh.f and shelxl.f is the first FORTRAN statement in which the array dimensions are specified by means of a PARAMETER statement; shelxh was compiled (using shelxlv.f etc.) exactly as described above for shelxl. Large versions of shelxs, shelxpro and shelxa may be created in the same way, but it is rather unlikely that they will ever be required. Further details are provided by comments in the respective sources.

SHELXL will print a suitable error message if it is necessary to increase the dimensions of the large arrays A or B. An additional warning sign is the 'maximum vector length' printed in the .lst file at the beginning of each refinement cycle; if it is too small (say less than 32) the program will still run, but with reduced efficiency. This applies to all computers but is especially serious on a vectorizing computer such as an older Cray or Convex.

A little care and fine-tuning is required so that such large structures can be refined efficiently. If the computer does not have enough physical memory available, or if the 'maximum vector length' is set too large, shelxh will run in disk exercising mode. This 'maximum vector length' refers to the number of reflections that are processed in one vector run, which may be smaller than the number in the input/output buffer. Some trial and error is needed to set the maximum allowed value so that the physical memory is fully exploited with a minimum of disk I/O for the virtual memory swap file. This number is set as the fourth parameter on the L.S. or CGLS instruction, and should be a multiple of 8; a good value to try for a 64MB computer is 64 (the third number on the L.S. or CGLS instruction is almost always zero). The array B is used as working space for these vectors (CGLS and L.S.) as well as for the least-squares matrix (L.S.). If the array B is not big enough, the program will use a smaller maximum vector run.

A further point to note for refinement of structures with more than 10000 atoms is that the SIMU and DELU instructions need to be broken up into several overlapping instructions, because the maximum number of atoms that can be referenced on any single instruction was arbitrarily set to 10000 (I never expected that this limit would be reached!).