DRAFT

Technical Note

Sloan Digital Sky Survey
Standards and Methods for Software Development

Don Petravick
Ruth Pordes
Fermilab

Robert Lupton
Princeton University

June 5, 1997

Abstract

This document describes the software standards and development methodology for the Data Systems for the Sloan Digital Sky Survey.

This document also describes the procedures for submission, acceptance, and integration of code into the production systems.

A Hypertext version of this document is available through the world-wide-web. Look for it through the SDSS home page

`http://fndauh.fnal.gov:8000/dss/doc/www/dss.home.html`

As this document was originally written in hypertext, it is advised it be accessed through this mechanism.

This documentation was prepared with LaTeX.

DRAFT printing: June 5, 1997.

Contents

Chapter 1

Introduction & Rationale

1.1 Scope

Achieving the science goals of the Sloan Digital Sky Survey (SDSS) relies on the development and use of a great deal of software. Much of the software used will not be written as part of the SDSS Project, but may already have been written, supported and maintained elsewhere. The software being developed by the collaboration itself will be developed and maintained by many people at many different institutions.

This document defines the current version of the software standards, methods and development toolkits for the core Data Systems for the Sloan Digital Sky Survey.

It is one of the charges of the Project Software Engineer in conjunction with the Science software Coordinator to research, define, propose, promulgate and arrange for review of these standards.

The standards, methods and toolkits will certainly evolve over the life of the project. This document is an attempt to collate and distribute those that are known now, as well as to raise issues of those that will be needed in the short term.

Clearly, the choice of a particular methodology or tool is always open to discussion. Sometimes the choice must be made between two technically equivalent alternatives; sometimes the choice is affected by non-technical reasons (e.g. cost, history etc); sometimes the choice is made to make best use of available resources and expertise. We make no statement here that these choices are technically the best possible. The goal is to define a set of standards and a methodology that meets the requirements of the survey and the collaboration, that is sufficiently benign as to be universally adopted and followed, that can be maintained, supported and extended with the available resources, and that can be expected to allow the needed evolution as the survey progresses - or allow programmatic conversion to new tools as the need arises.

1.2 SDSS Software

Software for the survey may be used in one or more environments:

- Survey Operation
  - TCC - 2.5m TCC, VMS Host Software, Microcontroller
  - MTCC - VMS Host Software, Microcontroller
  - Misc Devices - Filter wheel ucode, etc
  - Online systems - Imaging Camera and Spectroscopy Online and Host Computer Systems, Monitor Telescope Online and Host computer System,
  - Plate Plugging System,
  - Spectrograph - Mechanical Part, Camera microprocessor, Imaging Camera, Camera Microprocessor
- Data Processing Pipelines - Photometric, Astrometric, Spectroscopic, Monitor Telescope and others.
- Survey Strategy
- Offline science analysis by collaborators during the survey - at Apache Point or any of the collaborating institutions.

Software is used in "production" if it is integrated into the core SDSS Data Systems. This software will be used to produce the final survey science results. Clearly the requirements on this software are the most stringent, and thus we concentrate here on the standards and methods for its production and maintainance.

Software is either application-specific or a common-component - in which case it may be embedded in many different applications. Most common-component software will be provided (but not necessarily developed) as part of the software "toolkits" from Fermilab, or as part of the data processing pipeline science code or as science analysis code.

Software is categorized as either science-sensitive or non-science sensitive. Science-sensitive software needs to meet additional or different standards (e.g. be executed on specific test data) than the latter. Since in practice most actual programs include contributions from both categories all standards will be defined in this document.

1.2.1 Survey Operation

For Survey Operation, almost all software is written to work in a specific hardware context. Excepting common code, software in Survey Operation will be written for and tested only on the computers and equipment deployed at Apache Point. Furthermore, the development of the systems for Survey Operation is done as a single integrated project.

Many of these systems are based on prior systems (e.g ARC 3.5 meter telescope, Fermilab Drift Scan Camera, etc). No one has budgeted time to make these codes compliant to a number of new tools.

Only the mandatory requirements need to be satisfied by this software. The Project Software Engineer and Science Software Coordinator will define and negotiate with each group on a mutually acceptable set of standards and methods. For example, the core daq and online systems being developed at Fermilab are being done with careful attention to code and version management, testing, documentation, and programming techniques.

1.2.2 Data Processing

Data Processing is made up of a number of programs (Pipelines) which must run in a coordinated fashion in order to process the recorded survey data to produce the science results of the survey.

Although each Pipeline may be regarded as a separate software entity, each must conform to interfacing standards in accessing - reading and writing - the main survey databases, calibrations etc; they must run in a coordinated way; and each must conform to some level of code review, acceptance, robustness, quality assurance and regression testing criteria.

Although some of these pipelines will be coded and maintained by a single person, at least the photometric pipeline will include routines written by many different people.

Since it is not possible to precisely predict how or where scientific effort will have to be applied to this processing code, all pipelines should be constructed to support distributed development.

As science module algorithms are developed the methodology must support integration of new and replacement of old routines into the pipelines.

1.2.2.1 The Production System

The production system is the use of the pipelines to analyze data taken at the mountain in support of Survey Operation and Survey Strategy; for example, to allow plate drilling for spectroscopy in the required amount of time. The production system will be run on dedicated computing equipment at Fermilab. Data taken at the mountain must be analyzed in a pre-determined, small amount of time. This puts additional requirements on the stringency of the criteria for development and inclusion of code used in these applications.

1.2.3 Survey Strategy

The Survey Strategy programs are special data analysis programs that must have excellent access to the survey databases and a suite of supporting software tools to achieve their goals.

The requirements on the code development and methods can be less stringent than those for the production system - but the quality and maintainability would benefit from the code being as consistent as possible with the rest of the survey software, and in being able to make use of software developed for the other data systems.

1.2.4 Science Analysis

For the purposes of this discussion this is restricted to that "offline analysis" done by the survey collaborators during the test year and data taking years of the survey. There are no requirements on such code, except in so far as it may be later proposed for incorporation as common application code or for use in one of the above data systems. Presumably science analysis applications can be developed to make use of the common code if desired e.g. to access the main survey databases.

1.3 Goals

From the above it is clear that software used as part of the SDSS Data Systems will be run on many different instruments and computers, at many different sites and for many years. It will be used in many different applications to serve the project goals. The quality of the science results obtainable from the survey imposes requirements on the standards and methods used in the development and use of this software.

Additional requirements are imposed due to the fact that the software (even individual programs) will include contributions from many different people, developing at many different geographically distinct sites. The goals of this document are to define standards, methods, mechanisms and supporting software tools for adoption by the collaboration, to allow the software to meet the survey requirements. Given the length of the project this will clearly be an iterative process, with changes in definition, tool-kit and scope over the years.

Publication and promulgation of software (and in the future interface) standards and development methods at the start of the project is being done in order to allow code being developed now to meet the requirements on the survey software.

The nature of the Photometric Pipeline is that it poses the most stringent requirements and provides the best testbed for defining and applying these standards and methods. Anything that works for it should be more than sufficient for the rest of the software being developed.

Chapter 2

Process for Controlling Source Code Quality

Having standards for our source code, how shall we ensure that they are taken into account while the software is being constructed? How much attention should be paid to the standards? Who is responsible for constructing the software to standard?

Standards for scientific correctness are not dealt with in this note. Software coding standards are reviewed by a subcommitte of the SAC.

The goals for the standards do not include the establishment of standards lawyers. The standards do enumerate the specific elements of code construction which are important to the long-term maintenance, portability and consistency of use for the survey's software. In this sense, the standards are provided to the experiment in the same spirit that a common source code library, subroutines, etc are provided for the survey's use.

2.1 Responsibility for Following the Standards

2.1.1 Primary responsibility

Each person who produces code for the survey is primarily responsible for seeing that it conforms to the standards. He/she must be familiar with the standards document, and try to apply the standards to the code produced for the survey. All bodies of science code have a coordinator. One duty of the coordinator is to have a broader understanding of the standards and to ensure the salient software product conforms to the standards. Pipeline coordinators are responsible for giving feedback on standards to those working on the pipeline, and have resources through the Project Software Engineer available to them to help understand the applicable standards.

2.1.2 Control

How will we review the code, to make sure that it is well-constructed (or at least constructed to standard)? How much effort should be put into this activity? Who provides the feedback as to whether the persons with primary responsibility are applying the standards? Who do they feed-back to?

Each product will have several releases before a production version is frozen for the five years of formal operations. The Project Software Engineer and/or pipeline coordinator, builds the releases and supports their distribution as appropriate. At this time, the pipeline coordinator supervises a more formal scrutiny of the code.

It is not anticipated that every line of code will be reviewed. Rather, the amount of scrutiny need only be sufficient to form an opinion of how well the standards are being met, and to ensure the survey's interests in code quality are being served.

In the case of grave defects, the integration coordinator may refuse to accept the release. (This refusal can, of course, be appealed up the organization chart). Minor defects may be fixed on the spot, or scheduled by the pipeline coordinator for fix in a subsequent release.

The results are fed back to the Science Software Coordinator as well as through the project's organization chart to the Project Software Engineer.

Chapter 3

Software Environments

Here are the definitions of the survey software environments for portable code. Several levels of supported platforms are defined, and you are told which software your computer system must have to be at a given level. Portable software for the survey is written to conform to one of these levels.

It is the policy of the CIAC (driven by limitations on resources) that the burden of porting the software environment to a new type of system will be borne by those wishing to use the new class of machine.

The portable software environments table summarizes the different software environments and their requirements.

3.1 The Documentation Environment

This environment allows SDSS collaborators to construct and view survey documentation. It consists of:

- a TeX/LaTeX installation
- the ability to print and/or view postscript
- the ability to view dvi documents
- a browser for the WWW/HTML (World-wide web/ Hypertext mark-up language).

3.2 Portable Execute-Only Environment

This environment allows collaborators to acquire executable and supporting files such as help files and TCL scripts and run them on their own machine. A command procedure is available to help check that this environment exists.

This requires that the local system be of a specified type. The supported platforms are:

- SGI IRIX V5.3 or higher
- Digital OSF1 V3.2.d or higher
- Linux V2.0

Any supported machine must have:

- The documentation environment.
- The Fermilab UPS suite of products installed on it, and an identified local UPS administrator who works a few hours a month to keep the system's software up-to-date. Information about the UPS product suite is available from the following documents -
  - UPS Product Documentation
  - UPD Product Documentation

  The file *fermi.cshrc* (part of the SYSTOOLS product) should be sourced by each user to set up the UPS product suite access.
- Alteration of the system configuration and parameters as required by the SDSS data systems; access to commands that require root privilege e.g. setgid.
- The following software (of a revision we specify) located in /usr/local/bin, and that the user has /usr/local/bin in their path.
  - perl V5.0 (SDSS standard scripting language)
  - dropit (for UPS support)
  - funame (UPS Support)

It is the responsibility of the local site to obtain and build perl. It is available via ftp from prep.ai.mit.edu. The survey-wide UPS administrator will assist the local UPS administrator in obtaining the UPS environment, and learning about it.

The software distributed by the survey to operate in this environment shall be linked in a way that does not depend on the presence of sharable libraries not normally supplied in the vanilla software distribution for the platform. (e.g . there should be no assumption that a shareable FORTRAN library exists, if one does not come with every such system).

3.3 Portable Execute-Only Environment For Software Using The Survey Database

The system must have the portable execute-only environment.

The survey databases are constructed with proprietary software, e.g The Objectivity object-oriented database system. Systems running this software have run-time support for Objectivity installed and the Fermi supplied schema product.

3.4 Portable ANSI C Environment

The Portable ANSI C Environment allows SDSS collaborators to compile ANSI C code, and link it with various SDSS software packages. Platforms supporting the portable ANSI C environment must have:

- All the software specified in the portable execute-only environment.
- All of the software associated with the Dervish product. Any recompilation and relinking must be done using a standard shell script which drives the Makefiles, called sdssmake, which is located in the bin area of the Dervish product. This shell script (and other supporting scripts also located in Dervish) enforces the rules governing the construction of the survey's source code.
- standard compilers and developer's software:

```
     Operating system:       C-compiler      F77 compiler
     -----------------       ----------      ------------
     IRIX                    Sgi v7          F77 v7
     OSF1                    C 4.2           F77 3.8
     Linux                   gcc  2.7.2      gcc 2.7.2
The tools use the system supplied make program.
```

The survey requires the additional product gccfix if gcc is your compiler. This product supplies true ANSI include files; Those supplied with gcc are not ANSI standard. Gccfix is available from Fermilab.

3.5 Portable C++ Environment

The portable C++ environment allows SDSS collaborators to compile C++ code, and link it with various SDSS software packages. Platforms supporting the portable ANSI C++ environment must have -

- All the software specified in the portable execute-only environment.
- All of the software associated with the Dervish product. Any recompilation and relinking must be done using a standard shell script which drives the Makefiles, called sdssmake, which is located in the bin area of the Dervish product. This shell script (and other supporting scripts also located in Dervish) enforces the rules governing the construction of the survey's source code.
- standard compilers and developer's software:

```
OS:     C++            C-compiler     F77 compiler
---     ---            ----------     ------------
IRIX    SGI V7         Sgi c-compiler SGI F77
OSF1    C++ v5.0       C 4.2          F77 3.8
Linux   gcc 2.7.2      gcc  2.7.2     g77 2.7.2
```

3.6 Portable C++ Environment For Software Using The Survey Database

This system must have the portable C++ Environment. The survey databases are constructed with proprietary software, the Objectivity object-oriented database system. Systems running this software must have compile-time support for Objectivity installed and conform to all the requirements of the portable C++ environment.

3.7 Survey Source Code Library Environment.

The survey source code environment requires the portable execute-only environment. In general, collaborators will obtain read-only access to the survey's source code. Write access to the repository is governed on a module-by-module basis. Additional requirements are:

- The Survey CVS-related software products.
- The check_access script installed in /usr/local/bin
- The disk resource for a local, "clone" CVS repository, with software CVS administrator.

Chapter 4

Software development – sdssmake and Makefiles

The following methods listed for Makefiles complements the way CVS works. The methodology also includes support for using the picky compiler flags which ensure portability of the source code we are constructing, and will reduce problems when it comes time to accept the code.

4.1 Makefiles

All Makefiles must have at least the following four targets: `all`, `clean`, `install`, and `default`. For the `all`, `clean` and `install` target, the Makefile performs appropriate actions, and invokes Makefiles with the same target in directories below it.

`install`
: The `install` target copies those files in the current directory tree to the actual product declared to UPS. The Makefile invokes the `install` target for Makefiles in directories below it. It should also make sure that the CVS subdirectories are not copied.

`clean`
: The `clean` target restores the directory to the state it would be in if the source had just been checked out from the source code library. The Makefile invokes the `clean` target for Makefiles in directories below it.

`all`
: The `all` target makes everything in that directory. The Makefile invokes the `all` target for Makefiles in directories below it.

`default`
: The `default` target must be the uppermost target in the Makefile. Its sole function

is to remind the user to invoke the Makefile through **sdssmake**, not directly.

Two other standard targets are defined for Makefiles: `tags` and `make`. These targets must be used where appropriate.

`tags`

The `tags` target invokes /usr/local/bin/etags. When run, **etags** will create a file linking routine names with the file in which they are located. This file can then be read by **emacs** and used to get to routine sources. Tags are not created by default, the programmer has to issues specific commands to get them. For some products (e.g. photo), the `tags` target also generates tags files for TCL scripts and C-TCL interface code, using the sdsstools utility tcl_tags.

`make`

The `make` target invokes **make_make**. The Makefile dependencies should be automatically generated using this target.

Makefiles must be sensitive to the macros set by sdssmake, discussed below.

4.2 The sdssmake Makefile driver

Sdssmake is a "sh" script which drives our Makefiles. It sets several macros for the Makefile, and can pass general arguments to make. The sh script sdssmake is currently packaged in the sdsstools product. It provides for picky compilation (strict ANSI C and C++). This helps implement portable software. It is policy that portable software written for the survey be compiled using sdssmake, and that legacy software be augmented so that the interfacing files (e.g. .h files) can be included by programs built by sdssmake with no warnings.

Sdssmake includes a help facility listing all the flags it supports. This facility can be accessed by issuing the sdssmake command with the *-help* switch.

Sdssmake also deals with some specifically Dervish compilation problems. For example, we support a *-nopgplot* switch to support platforms without either f77 compiler or libraries. (Pgplot is our sole FORTRAN code in the portable ANSI-C Environment).

4.2.1 Standard Macros

4.2.1.1 The CC Macro

¡DD¿ The CC Macro is set to the requested compiler (cc, acc or gcc) along with a switch indicating the optimize or debug level for the compiler. Optimization and/or debugging is selected by the presence of *-optimise* and *-debug* flags in the invocation of sdssmake. Acc is the default compiler for SunOS, cc for IRIX and OSF1, gcc is the default for Linux. The user may select gcc for SunOS by invoking sdssmake with the *-cc=gcc* switch. A switch for ANSI C and C++ is available as well.

4.2.1.2 The CCCHK Macro

¡DD¿ The CCCHK Macro is passed to make, giving picky compilation flags for the particular compilation environment. These differ for each environment: *-wall* is selected for gcc, *-fullwarn* for IRIX, *-Xc -vc* for acc, etc. These flags are passed in a separate macro so that we can migrate a given product from not-up-to-picky to picky, file-by-file.

Within the Makefile, the default .c.o rule is overridden to be

```
$(CC) -c $(CCCHK) $(CFLAGS) $*.c
```

making picky compilation the default.

4.2.1.3 The LLIBS Macro

¡DD¿ The LLIBS Macro gives link libraries for C code. The FLIBS macro gives link libraries for FORTRAN code. The FORTRAN libraries are necessary because we link using the cc driver, not the f77 driver.

4.2.1.4 Other Macros

¡DD¿ Many other macros, including SHELL, AR, ARFLAGS and RANLIB are set by sdssmakeas well. (see the sdssmake documentation for more information)

4.2.2 Use of UPS

Sdssmake does NOT invoke the UPS setup command. This is forbidden. It is standard to have UPS on the system, and it is proper to use the UPS BUILD dependencies to set up the environment.

4.2.3 make_make

The make_make script uses the *-M* switch on the C compiler to generate the dependencies of .c files on .h files. The output is standard enough to be used with any compiler. Makefiles have a standard comment in them,

```
#START_DEPEND
```

lines below this are generated by make_make. Before being placed into the Makefile, the output from cc is massaged in two important ways: 1) Make_make discards any of the ANSI or system include files. These are not likely to change, and the path to these files will change depending on the compilation environment. Excluding these files adds the important property that the Makefile need not be rebuilt for each invocation. 2) Make_make converts any absolute path names for our dependent products into the a form using environmental variables. That is, /usr/products/SunOS/ftcl/include is converted to $(FTCL_DIR)/include.

The resulting dependencies are platform-independent. Make_make is invoked through the standard Makefile target "make:". This target need only be present for Makefiles which deal in C language code. We do not invoke it automatically, the programmers invoke it as required, and the Makefile just gets put back in RCVS.

4.2.4 filterWarn

Given a compilation environment, filterWarn catches standard errors from make, and possibly suppresses certain specific warnings. This is useful when compiling with all picky flags, and certain very specific warnings need to be suppressed. We then derive the benefits of picky compilation AND have every warning that eventually emerges to be a salient warning.

filterWarn is a perl script, which assumes that perl is in /usr/local/bin.

4.2.5 Other flags

sdssmake takes some other switches:

-filter indicates that output from the make is to be filtered with **filterWarn**. The specified *filterFile* is used in addition to the default filter file"

`${SDSSTOOLS_DIR}/bin/filterWarn.dat`

to control the filtering out of messages which are "spurious" rather than informative. filterWarn

-cflags *str* ¡DD¿ Additional string to be passed to the C compiler via the Makefile. This string is passed in the $(CFLAGS) variable.

¡/DL¿ Many other switches are available and described in the sdssmake documentation.

Chapter 5

Standards for ANSI C

5.1 Standards for ANSI C software

ANSI C software falls into one of three categories

- Portable Survey Software is written for the SDSS and runs in the portable ANSI C environment.
- Specialized Software runs on some environment other than the portable ANSI C environment, and is generally dedicated to some special use, like controlling the DAQ.

Portable Legacy Software is software adopted by the survey, but not written expressly for it. It may not be written in ANSI C, but it must run in the portable ANSI C environment. The Numerical Libraries section provides extra information about numerical libraries.

5.2 Portable Survey Software

All Portable ANSI C software written for the survey should compile without warnings in the standard survey compilation environment, on each supported platform. (Recall that the sdssmake system is part of this environment, and enforces rules about prototypes, etc.) The software should be packaged using the standard directory tree, and kept in the source code repository.

Template .c and .h files are available.

5.2.1 Compilation Environment

The standard compilation environment (including sdssmake) is currently provided with the sdsstools package.

5.2.2 Style

The Document "Recommended C Style and Coding Standards" (Henry Spencer et al) defines the base C style for ANSI C software, except as noted below:

- All names seen by the linker begin with a name which identifies the software product which owns them. For example Dervish external names begin with sh.
- Upper case, not underscore is used to delimit words within names. All names begin with a lower case letter: e.g. regReadAsFits.
- Procedure interfaces names are constructed according to the following pattern: product-identifier Thing [subthing] verb [preposition noun] e.g. shRegNew, shRegPixGetAsDbl
- Procedures which are not part of the public interface of a package, but cannot be declared to be static should follow the above rule, but begin with the prefix p_ , e.g p_shGetInternal
- Recommendations on Makefiles are superseded by those implemented in the standard compilation environment.

5.2.3 Restrictions on ANSI C

Portable ANSI C Software should be coded so that it will compile in a C++ compiler. In practice, this means to avoid using any of the keywords which are present in C++, but no ANSI C, for example, new and template.

All functions shall be prototyped in a header file (a .h file) or are declared static.

It shall be possible to compile all ANSI-C software all portable software in the sdssmake environment without warning.

5.3 Portable Legacy Software

C language software obtained for use in the survey, but not written explicitly for it, must still compile and run correctly on all the supported platforms. If the software is intended to be linked into other programs, then it must provide an interface which is strict ANSI C, as defined above. If the software is not compile-able without warnings in a C++ compiler, then its header files should allow for it being called as C-code from C++, e.g. use the linkage specification feature of the C++ language, properly conditionalized with cpp.

5.4 Specialized Software

Specialized software, like the kernel of the DAQ is not bound by this standard, but are bound by the general conditions given in the introduction.

Chapter 6

Standards

The supported assumptions about binary data types used in Survey software, beyond the assumptions listed in language standards, are specified here.

6.1 Bit Ordering

Code shall run on "big-endian" and "little-endian" machines. Signed integral types are implemented as two's compliment integers.

6.2 The Range of Integer Types

¡TABLE BORDER=1 CELLSPACING=0 CELLPADDING=3¿ ¡TR¿¡TH ALIGN=CENTER COLSPAN=4¿Primitive Data Type Value Ranges¡/TH¿¡/TR¿ ¡TR¿¡TH ALIGN=CENTER¿Language¡/TH¿¡TH ALIGN=CENTER¿Type¡/TH¿ ¡TH ALIGN=CENTER COLSPAN=2¿Value Range¡/TH¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=RIGHT¿unsigned char¡/TD¿ ¡TD ALIGN=RIGHT¿0..255¡/TD¿ ¡TD¿*¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=RIGHT¿signed char¡/TD¿ ¡TD ALIGN=RIGHT¿-128..127¡/TD¿ ¡TD¿*¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=RIGHT¿unsigned short¡/TD¿ ¡TD ALIGN=RIGHT¿0..65535¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=RIGHT¿short¡/TD¿ ¡TD ALIGN=RIGHT¿-32768..32767¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=RIGHT¿unsigned int¡/TD¿ ¡TD ALIGN=RIGHT¿0..4294967295¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=RIGHT¿int¡/TD¿ ¡TD ALIGN=RIGHT¿-2147483647..2147483647¡/TD¿¡/TR¿ ¡/TABLE¿

- The ANSI C `char` data type comes in three varieties: `signed`, `unsigned`, and plain. All three occupy the same amount of storage. The `signed` and `unsigned chars` respresent integers. The plain `char`, under ANSI C, can represent an integer or a character type.

The Survey limits the plain `char` type to represent character data rather numeric data; the `signed` or `unsigned` keyword should be used explicitly to represent an integer.

¡TABLE BORDER=1 CELLSPACING=0 CELLPADDING=3¿ ¡TR¿¡TH ALIGN=CENTER COLSPAN=5¿Languange Data Type Equivalents¡/TH¿¡/TR¿ ¡TR¿¡TH ALIGN=CENTER¿Language¡/TH¿ ¡TH ALIGN=CENTER¿Type¡/TH¿ ¡TH¿¡/TH¿ ¡TH ALIGN=CENTER¿Languange¡/TH¿¡TH ALIGN=CENTER¿Type¡/TH¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿FORTRAN 77¡/TD¿ ¡TD ALIGN=LEFT¿INTEGER*2¡/TD¿ ¡TD¿¡/TD¿ ¡TD ALIGN=LEFT¿C/C++¡/TD¿¡TD ALIGN=RIGHT¿short¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿FORTRAN 77¡/TD¿ ¡TD ALIGN=LEFT¿INTEGER*4¡/TD¿ ¡TD¿¡/TD¿ ¡TD ALIGN=LEFT¿C/C++¡/TD¿¡TD ALIGN=RIGHT¿int¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿FORTRAN 77¡/TD¿ ¡TD ALIGN=LEFT¿REAL¡/TD¿ ¡TD¿¡/TD¿ ¡TD ALIGN=LEFT¿C/C++¡/TD¿¡TD ALIGN=RIGHT¿float¡/TD¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿FORTRAN 77¡/TD¿ ¡TD ALIGN=LEFT¿DOUBLE PRECISION¡/TD¿ ¡TD¿¡/TD¿ ¡TD ALIGN=LEFT¿C/C++¡/TD¿¡TD ALIGN=RIGHT¿double¡/TD¿¡/TR¿ ¡/TABLE¿

6.3 Floating-Point Formats

Floating point numbers shall be in IEEE Std 754 format (*IEEE Standard for Binary Floating-Point Arithmetic*).

¡TABLE BORDER=1 CELLSPACING=0 CELLPADDING=3¿ ¡TR¿¡TH ALIGN=CENTER¿Language¡/TH¿¡TH ALIGN=CENTER¿Type¡/TH¿ ¡TH ALIGN=CENTER¿IEEE Type¡/TH¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=LEFT¿float¡/TD¿ ¡TD ALIGN=RIGHT¿single precision (32-bit)¡/TH¿¡/TR¿ ¡TR¿¡TD ALIGN=LEFT¿C/C++¡/TD¿ ¡TD ALIGN=LEFT¿double¡/TD¿ ¡TD ALIGN=RIGHT¿double precision (64-bit)¡/TH¿¡/TR¿ ¡/TABLE¿

- The C/C++ "double" can represent exactly all numbers in the C language types "int" and "unsigned int".

Various floating-point numbers are *not* to be generated or used by Survey code:

- NaNs (Not-a-Numbers), both signalling and non-signalling
- Denormalized numbers

The **sdssmake** script used to build Survey software attempts to set compilation flags to enforce these restrictions. These restrictions may not be uniformly applied to all platforms, therefore an emphasis should be made to avoid these values.

Users should also be aware that the FITS Standard does not restrict Tables from from containing NaNs and denormalized numbers. Survey software, namely **Dervish**, does permit these values to be read. Their use should be avoided (this includes, for example, casting or converting within C or C++) and, if necessary, they can be tested for with **Dervish** routines.

Chapter 7

Numerical Libraries

7.1 Numerical Software

This note outlines standards for common numerical software for the survey's portable software. Like all portable legacy software, numerical software should be called from C without the user being aware that the routines are in any way 'special'.

- Arrays must be 0-indexed.
- Matrices must not need to be transposed before use.
- Input-only scalar variables should be passed by value.
- External names used by these packages may not clash with other external names.
- ANSI C Prototypes shall provide for the routines's C bindings.

7.2 Commercial packages

Numerical packages whose use requires a fee shall not be used in portable software unless the CIAC agrees to add the package to the appropriate portable environment.

7.2.1 Numerical Recipes Software

Note that the Numerical Recipes in C software falls far short of these standards. This software may not be used in pipelines, as there is no agreement that each institution be properly licensed to use this software, and the Collaboration has no license to distribute it. Furthermore, there is no centralized effort to clean them up. Each person using these routines would have to be careful to copy them into their software project, ensure that all routines names are preceded by their product's unique prefix. This will at least prevent name conflicts.

7.3 Common Numerical software

In the Fall 1993 CIAC meeting it was agreed that the CERN program library numerical software be made available to the survey. This software is available to the survey, and may be re-used by collaborators on very favorable terms as long as the application is non-military.

While this is written in FORTRAN. Suitable bindings must be constructed before use by ANSI-C and C++ language programs. Common Bindings are in the SDSSmath product.

Chapter 8

Standards for C++

8.1 Basic Standards Document

Except as noted, standards for c++ shall be as in "Programming in c++ Rules and Recommendations" Ellemtel Telecommunications Systems Laboratories.

8.1.1 Exceptions

- C++ source files have the extension .cc
- C++ header files have the extension .h
- Inline functions need not be placed in separate files.

8.2 Database access

Because of the database tool chosen for the survey, c++ is the language of choice for database access. The following standards apply to c++ code related to the survey databases.

8.2.1 The SDB product

Software that follows the standards should use the survey access routines to interface with the survey databases.

8.2.2 Schema

C++ classes form the schema for our database. These classes should be stored in the schema modules in the survey CVS repository.

Chapter 9

Standards for FORTRAN

9.1 Standards for FORTRAN software

Here we give rules for dealing with FORTRAN software. There is no desire in the collaboration to write in pure F77, so we define an idiom of FORTRAN which adds a few extensions to F77 which seem necessary to our work, and which seem likely to persist in practical compilers. We take this position because there is no desire among scientists to use FORTRAN 90.

9.1.1 Taxonomy of Survey FORTRAN Software

Our FORTRAN software falls into one of three categories:

- Portable Survey Software is written for the SDSS and runs linked into applications in the portable ANSI C environment.
- Specialized Software runs in some environment other than the portable ANSI C environment, and is generally dedicated to some special use, like controlling the telescope.
- Portable Legacy Software is software adopted by the survey, but not written expressly for it. It need not be written in our idiom of FORTRAN, but it must run in the portable ANSI C environment. Software which is extensively modified under the supervision of a pipeline coordinator may not fall into this category without approval of the Project Software Engineer and Science Software Coordinator.

9.2 Portable Survey FORTRAN Software

All portable FORTRAN software written for the survey shall compile without warnings for each standard survey compilation environment. The sdssmake system is part of this environment and must be used to compile the code. The software shall be packaged using the standard directory tree, and kept in the survey's source code repository.

The compilation environment shall support FORTRAN source which includes embedded C pre-processor directives. Such code will have the file extension .F. FORTRAN software having common blocks or shared parameters must be implemented in .F files, and use the #include mechanism to share these definitions.

Portable Survey FORTRAN Software consists of packages of subroutines which are linked into C applications. This means that in addition to the FORTRAN code itself. It is necessary to provide:

- ANSI C headers, written to the survey's standards for Portable C Software for the package's public interface.
- An interface layer, written to the survey's standards for Portable C, which presents a FORTRAN package to the C and C++ programmer. This software shall implement the following features:
  - Input-only parameters of scalar type shall be passed by value into the interface layer.
  - The interace layer shall hide any hidden string length arguments. FORTRAN compilers in the current compilation environments may add an string length argument for every string in their formal parameter list. This is visible to the C programmer, and invisible to the FORTRAN programmer. The interface layer shall hide this from the C programmer.
  - The input layer shall hide the mapping of FORTRAN subroutines names onto C function names. FORTRAN compilers in the current compilation environments lowercase the FORTRAN procedure names, appending an underscore to them.
  - Readers are reminded that this layer shall be compile-able in a C++ compiler. An example in dervish is constructed to illustrate the proper techniques.
  - The interface layer shall hide common blocks from the C programmer.
  - The interface layer will accept arguments whose base types correspond to C base types of short, int, float, double and string.
  - Useful multiply-dimensioned arrays of a base have fundamentally different implementations in C and FORTRAN. The interface layer shall be carefully documented as the which style of multiply-dimensioned arrays are acceptable.
- Header files, with extension .h which define parameters in the public interface of the package so that they are available to C code in .c files and FORTRAN code in .F files.

9.2.1 Conventions

Programmers may assume that the INTEGER type refers to 32-bit two's compliment representation. Programmers may assume that REAL refers to the 32-bit floating point number native to their machine, and that DOUBLE PRECISION refers to the 64-Bit precision number on the local machine. INTEGER*4, an allowed extension, is preferred to INTEGER*2. It is alloweable to handle arrays of 16-bit unsigned data as INTEGER*2. The programmer shall document this carefully and use a function interface to manipulate individual numbers.

9.2.2 Detailed specification of the portable FORTRAN idiom

The Survey FORTRAN idiom is ANSI F77, except as modified below.

9.2.2.1 Allowed extensions

- Long identifiers which can be collapsed to unique seven character identifiers within a source file and files it includes. Collapsing is to be done by taking the first two and last 5 characters of an identifier
- The C pre-processor directive #include.
- The C-pre-processor directive #ifdef may be used within headers. The pre-processor symbol F77 will be defined while pre-processing survey FORTRAN. This symbol will be undefined while pre-processing for C. This allows parameters in the C-FORTRAN interface of a package to be defined in one file.
- INTEGER*2 and INTEGER*4 declarations.
- Handling arrays of unsigned short data as INTEGER*2, handling such numbers through a procedure call interface.

9.2.2.2 Explicitly not allowed

Valid in F77, but not allowed by the survey:

- Use of the blank common.
- IMPLICIT statements. Modern compilers have switches which have the same effect as the non-standard IMPLICIT NONE statement. However, IMPLICIT statements in the source override these switches. Eliminating these statements allows this superior method to be used.
- Use of the Dollar sign in identifiers.
- Use of I/O in subroutine packages, except writes to character string variables, as permitted by the standard. I/O is to be done from the framework.
- COMPLEX and LOGICAL types in a package's public interface. The interface should be specified using types in ANSI C.
- Common Blocks in the public interface.
- Pseudo-dimensioned arrays – It shall be possible to debug and verify code using the bounds checking features of a compiler.

Readers are reminded that the following are not allowed, because they have not been mentioned above, and are NOT part of F77

- Use of END DO
- Mid line comments, delimited by an exclamation point.
- The leading tab convention for source lines.
- Source lines longer that 72 characters.
- Use of Dynamic memory – Your caller must supply scratch space, if required.
- Structures

9.3 Portable Legacy Software

FORTRAN language software obtained for use in the survey, but not written explicitly for it, must still compile and run correctly on all the supported platforms. If the software is intended to be linked into other programs, then it must provide an interface which is strict ANSI C, as defined above. If the software is not compile-able without warnings in a C++ compiler, then its header files should allow for it being called as C-code from C++, e.g. use the linkage specification feature of the C++ language, properly conditionalized with cpp.

9.4 Specialized Software

Specialized software, like the Telescope Control program is not bound by this standard, but are bound by the general conditions given in the introduction.

Chapter 10

Software Packaging

10.1 Product Packaging

A software product is a collection of directories seen by a user. For example, the Libfits product provides an archive file which may be linked into programs, and a header file giving function prototypes, constants, etc. The survey will accumulate many such products. It is desirable that the physical layout of the products be uniform, so that one will always find a given type of file in a given directory.

10.1.1 Finding a product on a given system

Once installed on a system, products shall be located using the environmental variable of the form PRODUCT_DIR. These environmental variables are managed using the UPS tool. A product's environment variable is defined when the product is "setup" - i.e. setup libfits. Software may not be built with the assumption that it will be installed in a particular directory, except that perl scripts can assume perl is installed in /usr/local/bin.

10.1.2 Survey Software

10.1.2.1 Directory Structure

The following directory structure is required for products which are written for the survey:

- .h and other include files are held under a top-level include directory.
- Documentation is held under a top-level doc directory.
- TCL Scripts are held under a top-level etc directory.
- Archive files are held under a top-level lib directory. If only one .a file is supplied, it is named libproduct.a. A given directory tree may not have .a files for different CPU families (e.g. MIPS and SPARC)

- Executable files and scripts directly invoked by the user are located in a top-level bin directory. A given directory tree may not have executable files for different CPU families (e.g. MIPS and SPARC)
- A top-level ups directory, containing the scripts setup.csh unsetup.csh, setup.sh, and unsetup.sh is supplied with the product.

10.1.2.2 Required files

Message codes: every product must define its numeric error codes in a file called product_msg_c.h (or product_msg_cc.h for C++) in the same style as DERVISH does.

10.1.3 Legacy Software

Software not written for the survey need not conform to the directory structure given above.

10.2 Directory Structure and Required Files for Development

Portable software constructed for the survey should be packaged according to the following rules.

10.2.1 Directory Structure

To the extent possible, the directory structure used for software during development should be the same as its packaging for delivery.

10.2.2 Makefiles

Each directory in a product's source tree has a Makefile. It MUST be named Makefile. The top level Directory in a product must have a Makefile. Makefiles must be constructed to survey standards.

10.2.3 Relationship of development area to CVS library

Each product's development area corresponds to an CVS module. The CVS module has the name of the product. The product's CVS Source code module may include other CVS source code Modules.

Chapter 11

Survey Source Code Repository

The survey provides a master source code repository, located at Fermilab, to hold much of the code used in the survey. The software, CVS, supports developers at sites remote to Fermilab.

The source code library is a tool for handling the survey's source code. As code is modified, it keeps the current copy, and saves the differences between the new version and the previous version. The library allows creation of branches, so that two or more different versions of the source can be maintained, and later merged.

Using the source code manager, you can look at the survey's source code, see what the code was like at any date, or in any versioned release. You can get at the source code world-wide.

Extensive documentation is available with the CVS installation.

11.1 Policy

11.1.1 Who must put their source into the library?

All survey source code shall use the master repository, except that the following types of software are excused from using the repository, though the repository is available, if the project leaders choose to use it.

- Software in the VME front ends of the data acquisition system.
- Software in the instruments.
- Software running on the 2.5m and monitor telescope TCC, and its peripheral computers.
- IRAF Distribution for the spectroscopic extraction.

11.1.2 Who can access the library?

11.1.2.1 Read Access

Survey collaborators may gain read access to the survey library from any computer which supports the survey source code library environment. Contact is through the the local ¡A href=administrators.html¿CVS Administrator for IAS, JHU, UC, APO, JPG, FNAL. Collaborators at other sites should contact the CVS Master Source Repository Administrator.

11.1.2.2 Write access

Mostly, each product is composed of substantially its own source code, and links to supporting libraries. Therefore, the bulk of what is really in a product is contained in a few CVS Modules, and the product coordinator decides who can write the source code library.

However, there are a very few CVS modules whose purpose is to allow the sharing of code as source, rather than link libraries. This sharing of code is to be implemented using CVS; the code itself should be placed in a distinct CVS module. The most important module of this sort is "schema", which holds the C++ classes which describe data in the archives. These kinds of modules have a coordinator who

- controls the main branch for each file
- makes sure that all who have write-access to the module understand it is special.

11.2 Miscellaneous Methods

11.2.1 Miscellaneous and Test Data

Several gigabytes of library space are planned. It is possible to save binaries in the library; therefore, one may save small amounts of binary test data that are versioned along with the source code using CVS.

11.2.2 Etiquette

The source code library gives a very good record of code's history. The following rules for managing the source help make that record understandable:

- Do not gratuitously re-format code. The substantive changes in code do not relate to the whitespace, but the other characters. Once code is re-formatted, it is more difficult to understand how it has changed. If you must re-format, check the source out, re-format it, but do not change it in any other way, and check it back in.

- Record the nature of modification to the software using the CVS commenting mechanism. Make sure that the comments fairly reflect the changes to the file. Do not be afraid of multi-line comments; use the pop-up editor feature of CVS.

11.3 Standards for Scripts

SDSS supports scripts written in either **shell**, **perl** or **TCL**.

Perl Notes:

- Survey systems are at Perl 5.¡/LI¿
- Perl executable is located in `/usr/local/bin` on each system.¡/LI¿
- Perl scripts are assumed to be interpreter files, so they should begin with the following line:
  `#!/usr/local/bin/perl` ¡/LI¿

Chapter 12

Methods and Standards for Pipeline Developers

12.1 Background

A pipeline is software which reduces the survey's primary data – tapes which come from the data acquisition system. One characteristic of pipelines is, given we know what is on the input tapes, we can pre-stage from the survey's databases all the supporting data required to reduce it.

12.1.1 Portability

It shall be possible for collaborators to run versions of any pipeline at their own institutions, provided that they have established the portable execute-only environment on their computer. This precludes direct use of the Survey's C++ databases.

The software in each pipeline shall be constructed in such a way that is shall be possible for collaborators to develop science modules providing they have established the Portable ANSI C Environment and the Survey Source Code Library Environment.

12.1.2 Production

The production environment, where the data will be reduced, is to be provided by Fermilab. The production machines are two AlphaServer 8200 Model 5/300's (cpu type: DEC 7000), each with 1 GB of memory, running Digital Unix. Dervish developers are responsible for architecting a software framework which will allow this reduction to proceed smoothly.

12.2 Standards

The above discussion gives the following:

- Pipeline modules may not use the survey database software directly.
- Pipeline modules may NOT be coded in C++, but should be coded in the ¡A href=ansic.html¿Survey's standards for ANSI C .
- Pipelines must run in the portable execute-only environment.
- All source code for pipelines (except as excused by the CIAC) must live in the Survey Source Code Library.
- Periodically, versions of the software which comply with the survey's standards must be identified in the source library, and made available to the collaboration at large.
- Separate jobs (called pre-processing and post-processing) shall muster and store data from/to the survey database. These modules should be written in C++. All access to the survey database will be through sdb.
- Except for cases of compelling software re-use (determined by the CIAC) pipelines shall be implemented using the Dervish tools.
- Each pipeline supplies test data which illustrate the correct functioning of the pipeline.
- Part of the interface to each pipeline are well-defined file formats for the output of the pre-processor, and input to the post-processor, allowing others to create test data and evaluate the pipeline.
- Software is accepted for distribution and production according to the survey's Process for Controlling Source Code Quality.

Chapter 13

Standards

We specify supported tapes, tape drives and format of tape.

13.1 Drives

Quantum DLT drives shall be used to exchange data with the Data Processing System. Users shall be equipped to receive tape data on DLT4000 cassette, hence shall purchase a DLT4000 compatible drive for their institution. Tapes may be written with compression turned on.

13.2 Tapes

DLT2000 tapes may be applied when appropriate. Note that DLT4000 drives read and write DLT2000 tapes. Tapes shall comply with the following:

13.2.1 Blocking

Data blocking visible to software shall be a multiple of four bytes in length. When FITS is used, the data blocks shall be a multiple of 2880 bytes, the FITS logical record size. No data block shall contain more than 144000 bytes. All data blocks shall contain more than 11 bytes. All data blocks shall contain a multiple of four bytes. Tapes written by the imaging camera data acquisition system shall be written with 144000 byte data blocks.

13.2.2 External (Written or Printed) Label

All Tapes that might enter the production system at the Feynmann Computing Center must have a preprinted label with the form PP-AA-NN where PP is a prefix preassigned by the computing division; AA is alphabetic; and NN is alpha-numeric. This label is called the volume serial number, VSN, and is used to request tape mounts at FCC.

Tapes initialized for APO shall have the prefix JL.

Tapes initialized for FNAL shall have the prefix JG

Tapes initialized elsewhere shall have the prefix JN.

No meaning shall be attached to the AA-NN fields.

DLT Cassettes shall be labeled by inserting a fiberboard blank into a detent. The detent shall hold a 2 7/32" x 13/16" label. Paper or sticky labels should not be substituted for the fiberboard stock. The DLT Cassettes should be kept in their original polyethylene case. It shall be possible to read the VSN through the polyethylene case. This means that the survey must write the 6 character Fermilab VSN on the fiberboard using a high-contrast ink.

13.2.3 Internal (written on the tape) labels

When ever an external label, as described above, is applied to a tape a ANSI VOL1 header shall be written on the tape using the program sdsstapeinit.

13.2.4 Tape Mount Procedures

At FCC, tape mounts shall be requested using the Operator Control Software, OCS, which is documented in the OCS Reference Guide, CD Doc# GA0012, available in the Computing Division Library or with the OCS product, setup OCS. OCS should verify the VOL1 header.

Applications shall inspect the FITS header to ensure the correct tape is being read.

Applications must always write to initially empty tapes. An empty tape is one that has a VOL1 label followed by a pair of file marks. A procedure will be available for verifying that a tape is empty.

After data is written to tapes a READ ONLY sticker must be attached and the tape must not be rewritten unless it is recycled in a controlled environment.

13.2.5 File Marks

The last data block on a tape should be followed by two tape marks. Files shall be separated by single tape marks. Files shall be shorter than 500 Mega-bytes.

Appendix A

Directory of Contacts

A.1 Survey-wide Administrators

- CVS Master source repository administrator:
  - Chih-Hao Huang (huangch@fnal.gov)
- Bug Reports for DERVISH product:
  - Eileen Berman (berman@fnal.gov)
  - Gary Sergey (sergey@fnal.gov)
- Survey-wide UPS administrator:
  - Philippe Canal (pcanal@fnal.gov)

A.2 Local Administrators

After becoming initially familiar with the products, these administration positions should not consume more than a 8-10 hours a month.

The local UPS administrator needs to distribute new releases of survey software from the distribution nodes at Fermilab as they become available. Automated support for this person will become available before 1994. This person needs to

- delete old versions of software when they are no longer useful,
- watch how full the disks are
- ensure that collaborators at the site have the required lines in their profile (or similar) files to run the setup software.
- Deal with the local system policy regarding new releases of compilers, privileges, etc.

The local CVS administrator needs to provide disk space for a local repository, and serves as a local contact point for users: mostly, this means helping users configure their .rhosts files, etc.

A.2.1 Fermilab OLS

- UPS Administrator Philippe Canal (pcanal@fnal.gov)
- CVS Administrator Chih-Hao Huang (huangch@fnal.gov)

A.2.2 Fermilab EAG

- UPS Administrator Chris Stoughton (stoughto@leon.fnal.gov)
- CVS Administrator Chris Stoughton (stoughto@leon.fnal.gov)

A.2.3 Apache Point Observatory

- UPS Administrator TBA
- CVS Administrator TBA

A.2.4 Japan Promotion Group

- UPS Administrator K. Shimasaku (bshimas@ghz.mtk.nao.ac.jp)
- CVS Administrator K. Shimasaku (bshimas@ghz.mtk.nao.ac.jp)

A.2.5 The Johns Hopkins University

- UPS Administrator A. Connolly (ajc@skysrv.pha.jhu.edu)
- CVS Administrator A. Connolly (ajc@skysrv.pha.jhu.edu)

A.2.6 Princeton University & Institute for Advanced Study

- UPS Administrator Robert Lupton (rhl@astro.princeton.edu)
- CVS Administrator Robert Lupton (rhl@astro.princeton.edu)

A.2.7 The University of Chicago

- UPS Administrator A. Meiksin (meiksin@mohawk.uchicago.edu)
- CVS Administrator A. Meiksin (meiksin@mohawk.uchicago.edu)

A.2.8 The University of Washington

- UPS Administrator TBA
- CVS Administrator TBA

Appendix B

./www/Makefile.html

```
#<HTML>
#<PRE>
#
# This is the Makefile in the standards area.
# It also serves as an example for the
# standards documentation. Note all four standard targets
# are present, even though the "all" target has nothing to do.
#
SHELL = /bin/sh

default:
@echo Please invoke this Makefile using SDSSMAKE

#Install to where the files will be seen from the dss home page.
install:
@if [ -d $(DSS_DIR)/doc/standards/www ]; then \
cp *.[hc] $(DSS_DIR)/doc/standards/www; \
cp Makefile $(DSS_DIR)/doc/standards/www; \
cp environmentCheck $(DSS_DIR)/doc/standards/www; \
cp *.html $(DSS_DIR)/doc/standards/www; \
else \
echo no DSS_DIR/doc/standards/www to install to; \
fi

#nothing to make in this directory (it contains only .html files! but
#the target is here anyway.
all:
;
```

```
#
# Scrub down to what would have been had from the source repository
#
clean :
@ $(SDSSTOOLS_DIR)/bin/clean
rm -f *.log *.toc  *.aux  core *.ps *.dvi
rm -f *~ core *.bak *.orig *.old .\#* \#*\#

#</PRE>
#<HTML>
```

Appendix C

./www/C-template.c

```
/***************************************************************************
****************************************************************************
** FILE:
**      myfile.c
**
** ABSTRACT:
**
** ENTRY POINT          SCOPE   DESCRIPTION
** ----------------------------------------------------------------------
** shGateNext public main interface to GATEs
** shGateInit public malloc the space for the gates
** shGateNew public create a gate
** shGateDel public delete a gate
** gateNew local local interface to shGateNew
**
** ENVIRONMENT:
**      ANSI C.
**
** REQUIRED PRODUCTS:
**
** AUTHORS:
**      Creation date:  May 7, 1993
**      George Jetson
**      Smokey Stover
**
****************************************************************************
***************************************************************************/
#include <stdio.h>
#include "dervish.h"
```

```
#include "C-template.h"

static void gateNew(int n, GATE **g); /* local interface to shGateNew */

/*============================================================================
**============================================================================
**
** LOCAL MACROS, DEFINITIONS, ETC.
**
**============================================================================
*/
/*----------------------------------------------------------------------------
**
** LOCAL DEFINITIONS
*/
static GATE *first_gate = NULL; /* first of set of GATEs */
static char *foo[SH_SYMBOLIC_NAMES]; /* names of foos involved in SDSS */
static void (*func)(int n, GATE **g); /* create a <n>-bar GATE */
static int gate_is_inited = 0; /* has shGateInit been called? */

/*-----------------------------------------------------------------------------
**
** GLOBAL VARIABLES
*/
#define NGLOAT 10 /* number of time to gloat at user */

/*=============================================================================
**=============================================================================
**
** ROUTINE: shGateNext
**
** DESCRIPTION:
** This function is the main interface to GATEs. It assumes that you
** have already called shGateInit and that it is not Tuesday (GMT).
**
**
** RETURN VALUES:
** g a gate
**
** CALLS TO:
** shError
** shAssert
**
** GLOBALS REFERENCED:
```

```
**
**=====================================================================
*/
GATE *shGateNext(void)
{
   int i;
   static int first = 1; /* is this the first call? */
   GATE *g; /* the next GATE available */

   if(first) {
      first = 0;
      if(!gate_is_inited) { /* they didn't call shGateInit */
 shError("shGateNext: You should call shGateInit first\n");
 for(i = 0;i < NGLOAT;i++) {
    shDebug(1,"Naughty Naughty");
 }
 exit(1); /* we could be nice and call shGateInit
   for them, but let's not */
      }
   }
/*
 * OK, we know that they initialised things. Proceed.
 *
 * look for a five-bar gate
*/
   shAssert(first_gate != NULL); /* this can't happen */
   g = first_gate; /* we know that this isn't NULL */
   while(g->bar != 5 && (g = g->next) != NULL) continue;

#if 0 /* removed while I try to think */
   if(g == NULL) { /* we failed to find one that we liked,
   so let's make one */
      (*func)(5,&g);
   }
#endif

   return(g);

}

/*=====================================================================
**=====================================================================
**
** ROUTINE: shGateInit
```

```
**
** DESCRIPTION:
** This function must be called before shGateNext as it mallocs the space
** that the rest of the world relies on
**
**
** RETURN VALUES:
**
** CALLS TO:
** shMalloc
** shError
**
** GLOBALS REFERENCED:
**
**=========================================================================
*/
void shGateInit(void )
{
   GATE *gates; /* malloced space for gates */
   int i;

   if(first_gate != NULL) {
      shError("shGateInit: you called me twice");
      return;
   }
/*
 * We need to allocate space for lots of gates. This is pretty
 * horrible code, but beautifully structured
 */
   gates = (GATE *)shMalloc(SH_NGATE*SH_NFIELD*sizeof(SH_NGATE));
   first_gate = &gates[0];

   for(i = 0;i < SH_NGATE*SH_NFIELD - 1;i++) {
      gates[i].bar = -1;
      gates[i].next = &gates[i + 1];
   }
   gates[i].next = NULL;
/*
 * we'll use gateNew via a function pointer just to confuse people
 */
   func = gateNew;
/*
 * now initialise foo; just place holders for now
 */
```

```
   for(i = 0;i < SH_SYMBOLIC_NAMES;i++) {
      foo[i] = "Rule Britannia";
   }
}   /* shGateNext */

/*=========================================================================
**=========================================================================
**
** ROUTINE: shGateNew
**
** DESCRIPTION:
** Create gates.
**
**
** RETURN VALUES:
** g a gate
**
** CALLS TO:
** shMalloc
**
** GLOBALS REFERENCED:
**
**=========================================================================
*/
GATE *shGateNew(void)
{

   GATE *g = shMalloc(sizeof(GATE));
   g->bar = 0;
   g->next = NULL;

   return(g);
}   /* shGateNew */

/*=========================================================================
**=========================================================================
**
** ROUTINE: shGateDel
**
** DESCRIPTION:
** Delete a GATE
**
** RETURN VALUES:
**
```

```
** CALLS TO:
** shFree
** shAssert
**
** GLOBALS REFERENCED:
**
**============================================================================
*/
void shGateDel
   (
   GATE *g
   )
{   /* shGateDel */

   shAssert(g->next == NULL);
   shFree((char *)g);

}   /* shGateDel */

/*============================================================================
**============================================================================
**
** ROUTINE: gateNew
**
** DESCRIPTION:
** Create and return an n-bar gate
**
** RETURN VALUES:
**
** CALLS TO:
** shGateNew
**
** GLOBALS REFERENCED:
**
**============================================================================
*/
static void gateNew
   (
   int n, /* IN: */
   GATE **g /* OUT: */
   )
{   /* gateNew */
   *g = shGateNew();
   (*g)->bar = n;
```

```
}   /* gateNew */
```

Appendix D

./www/C-template.h

```
/**************************************************************************
*+
** FILE:
** C-template.h
**
** ABSTRACT:
** This file contains all necessary definitions for the GATEs
**
** ENVIRONMENT:
** ANSI C.
**
** AUTHOR:
** Creation date: May 07, 1993
**
*/
#if !defined(C_TEMPLATE_H)
#define C_TEMPLATE_H
/*
 * Define all public information about GATEs, which are the primary data
 * structure used in my attempt to control the sdssBisonHerd
 */
#define SH_NFIELD 30 /* as many fields as we can handle */
#define SH_NGATE 10 /* max number of gates per field */
#define SH_SYMBOLIC_NAMES 100 /* max number of names of animals */

typedef struct gate { /* describe a gate */
   struct gate *next; /* next gate in the list */
   int bar; /* number of bars on the gate */
} GATE;
```

```
typedef enum { BARBED, SHINY, RUSTY } SH_WIRE; /* types of wire expected */

GATE *shGateNew(void); /* make a new GATE */
void shGateDel(GATE *g); /* destroy a GATE */

#endif
```

Appendix E

./www/environmentCheck

```
#!/bin/sh
#
# This script checks for the presence of the SDSS portable execute-only
# environment
#
# 12-August-1993, DLP
#

in_order=T
echo "looking for portable-executue-only environment";

# Standard directory, executables, and it is it in your path?
if [ ! -d /usr/local/bin ] ; then
   echo "the standard directory /usr/local/bin is missing" >&2 ;
   in_order=F
fi;

for f in perl dropit funame  ; do
   if [ !  -f /usr/local/bin/$f ] ; then
      echo "the standard file /usr/local/bin/$f is missing" >&2 ;
      in_order=F
   fi;
done;

if [ `echo $PATH | sed  -e 's|/usr/local/bin||'` = $PATH ] ; then
   echo "/usr/local/bin is not in your path" >&2
   in_order=F
fi;
```

```
osname=`uname`
if [ $osname != SunOS -a $osname != IRIX -a  $osname != OSF1 -a  $osname != Linux ] ; then
   echo "Sorry, this is not a supported operating system" >&2
   in_order=F
fi;

#
# Check to see if UPS (UNIX Product Support) is present
# if this test fails, then either your account is mis-configured
# or your machine is mis-configured (probably not true if you passed
# the above tests.
#
if [ "$PRODUCTS" = "" ] ; then
   echo "products database not found or does not exist" >&2
   in_order=F
fi;

#
#
#
if [ $in_order = F ] ; then
   echo "Sorry, Either your system or your account is mis-configured"
   exit 1;
   else
   echo "I think your system is at least at the execute-only level."
fi;
```

Portable Software Environments

	(1) Doc	(2) Exe	(3) Exe & Db	(4) Ansi C	(5) C++	(6) C++ & Db	(7) Rcvs
TeX/LaTex	X	X	X	X	X	X	X
Postscript	X	X	X	X	X	X	X
WWW	X	X	X	X	X	X	X
IRIX or SunOS		X	X	X	X	X	X
UPS suite & administrator		X	X	X	X	X	X
Root System Control		X	X	X	X	X	X
perl, dropit & finame in /usr/local/bin		X	X	X	X	X	X
Versant Run Time Support			X			X	
SHIVA Software				X	X	X	
C & F77				X			
C++ & F77					X	X	
RCVS Related Software							X

Figure E.1: Table 1