Retallack

SEP RECEIVED 24

National Radio Astronomy Observatory
Very Large Array

15th November 1982

To: Addressee

From: Bob Duquet Bob Duquet

Subject: A FORTRAN Standard Commands Package

The attached document describes the results of work that I have been doing during the past two months. It was, essentially, a feasibility study whose purpose was to estimate the amount of effort that would be required to replicate the Standard Commands package in FORTRAN 77.

The were several reasons why such a study was thought to be worthwhile. First, we need to maintain (actually, to re-establish) a standard interface between our users and all the software that they need. That software is now being dispersed across many machine types but the original user interface was implemented in a language (SAIL) that is available on only one machine.

A related reason is the likelihood that some day we will want to migrate from the DEC10 to newer hardware. Even now, we ought to be able to evaluate alternatives such as the Jupiter or CRAY or HEP machines without the distraction of a machine-specific software library. For that reason a committee of two (Don Wells and I) has been charged with evaluating present and foreseeable programming language options. This study can be considered a part of that committee work.

Another reason for this study was to see if our software productivity could be improved. The SAIL Standard Command package, as elegant as it is in concept, is nevertheless the foremost reason why maintenance of our production programs is so painfully difficult. The problem lies in an abuse of SAIL's high-level macro facility. The very deep nesting of these macros in the Standards Command package has two ill effects: it requires that all Standard Command code be recompiled for each modification to each program no matter how trivial the program or how slight the modification; even more devastatingly, it translates the program text into a pile of mush that is unrecognizable by the author.

The initial plan was to use the SORTER machine for this study. PDP11 software includes a FORTRAN 77 compiler, the SORTER machine was available for software development and, of course, the most immediate use for a FORTRAN Standard Commands package would be in conjunction with pipeline software which is expected to run on the

SORTER machine. (Even though much of the relevant software has already been written without the benefit of such a package.)

Within a few days of starting the project, the SORTER machine developed hardware problems that kept it out of service for the next 6 weeks. In what turned out to be a rather fortuitous move, the programming that was involved in this project was transfered to a VAX (VAX3) just at the time the SORTER troubles started. The move was doubly advantageous because the VAX implementation of FORTRAN 77 is the full ANSI standard whereas the FORTRAN 77 available on the PDP11 is merely the ANSI subset language with a few DEC-specific (non-standard) enhancements.

The study was conducted by actually trying to write at least part of a Standard Commands package for the VAX. The programming was continued only far enough to be a proving ground for ideas about the package structure. It also demonstrated the applicability and usefulness of FORTRAN 77 features that were not available in earlier versions of the language.

Here is a summary of the results of this study:

a) The feasibility of the project has certainly been demonstrated. In addition, the study has demonstrated the feasibility of including in the package certain enhancements that were suggested.

b) A set of specifications for a FORTRAN Standard Commands package has been developed and is being circulated herewith for suggestions and comments.

c) A partially completed FORTRAN version of the Standard Commands package now runs on a VAX. It should be relatively easy to modify those programs so that they conform to whatever final specifications emerge from discussion of item b above. They therefore represent a large first step taken toward achievement of a fully useable implementation.

d) A program that served as a test bed for the above can now serve as a working model of how the (existing version of the) package looks to a user. It will also reveal details of the programming steps that must be taken in order to use the package for a specific application.

e) The experience gained during this study has been used as the basis for estimates of the time and effort required to create a complete FORTRAN package for each of the machines we presently use. With somewhat less confidence, we have also prepared estimates of the time and effort that would be needed to incorporate these packages into the present library of user-accessible software.

FEASIBILITY OF A FORTRAN STANDARD COMMANDS PACKAGE

Robert T. Duquet

November 1982

PART 1: INTRODUCTION - A summary of what was done and why.

PART 2: PACKAGE SPECIFICATIONS - A distillation of what seems to be both feasible and desirable. This portion should be examined carefully by anyone who has a stake in what the final product (if any) will look like.

PART 3: DESCRIPTION OF THE EXPERIMENTAL PACKAGE - Describes an existing prototype package. This portion emphasizes the structure of the package and the programming techniques that were used in writing it.

PART 4: DOCUMENTATION OF THE EXPERIMENTAL PACKAGE - This is mostly a collection of the comment lines abstracted from various routines of the experimental package. It is aimed primarily at those who might be interested in the internal workings of the package.

PART 5: UTILITY ROUTINES - Basically a continuation of the previous section, Part 5 describes a set of routines written for this study but of wider usefulness.

PART 6: THE "TEST BED" PROGRAM - Shows the full text of a small program that exercises the package. It is included here for those who would like a paradigm of package utilization.

PART 7: ESTIMATES OF TIME AND EFFORT - This is the payoff. These estimates should help those who must decide whether or not a completely operational FORTRAN version of the Standard Command package is worth the effort required to write it.

Part 1: INTRODUCTION

Until recently, all of the software needed by our users consisted of DEC10 programs that incorporated a standard user interface - the Standard Commands package. The relatively few PDP11 programs were driven by users through standard-conforming DEC10 programs.

The advent of the AIPS system on the VAX, with which the user interacts directly, has changed this situation; it requires familiarity with a different user interface. Since our users will also be required to interact directly with the new "pipeline" software now being written for the PDP11s, there is a real possibility that yet another user interface will be created.

This trend can be reversed only if the programming tools that provide the user interface are made machine-independant. This report describes work done to evaluate the feasibility of replicating the DEC10 Standard Command package in FORTRAN 77.

The overall question of feasibility involved four parts. One was the extent to which the SAIL Standard Commands package utilizes TOPS10 system services. If the dependency were large, differences between operating systems might make it difficult to replicate the DEC10 package on other machines.

A second source of problems might be the absence of recursion in FORTRAN. (The ANSI committee for FORTRAN is currently debating the suggestion that recursion be allowed in FORTRAN 8X; it is not allowed in FORTRAN 77.) Since SAIL does allow recursion, a great deal of effort might have been required to reformulate recursion-based algorithms used in the SAIL package.

Facilities for handling character data have been notoriously absent from previous versions of FORTRAN; in SAIL they are excellent. Since the Standard Commands package accepts free form input which must be parsed according to fairly complex rules, the string-handling features of FORTRAN 77 needed to be evaluated for this application.

Finally, it was anticipated that communication between FORTRAN subprograms would be awkward compared to the automatic communication that exists when high-level macros blend Standard Command variables and user code into one monolithic program.

It turns out that none of these difficulties was particularly severe.

With a few minor exceptions, the SAIL Standard Commands package does NOT use a large number system-specific services. Typical exceptions are: capturing the compilation date and time (for run time documentation) and ascertaining the identity of the current user (so that file names can be constructed appropriately).

The only significant use of recursion was found in the NEEDNUM routine where recognition of the allowable date and time formats involves identification of numerous subfields. In the SAIL package it was convenient to use the same code to parse the total field and its subfields i.e. to use recursion. This is no big deal however. For one thing, the depth of recursion is very limited and known in advance; for another, it is questionable design to include in a single procedure a basic service function (that of identifying the units of numeric data) and a higher level task (parsing a complex string that consists mostly of non-numeric characters). In other words, splitting the analysis of date and time into a separate non-recursive procedure for FORTRAN, is probably an improvement.

The CHARACTER variables of FORTRAN 77 are certainly not as convenient to use as the STRING variables of SAIL. The most important difference, of course, is the fixed pre-determined length of the FORTRAN variables. (Dynamic assignment of CHARACTER variable storage, a non-standard extension of FORTRAN 77, should be avoided like the plague!) On the whole, the CHARACTER-handling features of FORTRAN 77 on the VAX were adequate for writing the Standard Command package. The situation on the PDP11, however, may be an entirely different story; it is precisely in this area that the subset of FORTRAN 77 differs most markedly from the full version of the language.

A FORTRAN Standard Commands package could be designed as a self-contained task that would communicate with other tasks through system-dependant task-task services. That was not considered an attractive possibility. It would severely limit the adaptability of the package to special program requirements.

FORTRAN provides several alternatives for communication between subprograms and programs that CALL them. Straightforward use of a COMMON area, in which all program parameters would be defined, was rejected. It would have meant recompiling the entire package for each change that affected the COMMON area (e.g. for the frequently required addition of special-purpose commands). Only two small COMMON areas were used in the experimental package; they define quantities that should change only infrequently, if ever. The chosen alternative, inter-modular communication via arguments, turned out to be quite adequate for the needs of the package.

In summary then, the study showed no special problem that would make it difficult to replicate the existing DEC10 Standard Command package in a more machine-independant language.

Part 2: PACKAGE SPECIFICATIONS

User Interface

Ideally, differences between the current DEC-10 Standard Command package of SAIL macros and a new FORTRAN version should all be "upward compatible". In other words, users should be able to continue using programs as they had been doing previously, without even being aware of changes that had been made. In practice, some improvements that have been suggested (such as a change in the appearance of the HELP text) would be fairly obvious to users. (Of course, changes that were sufficiently popular could be "retrofitted" to the older SAIL package.)

Some of the proposed changes are:

i) The response to the HELP command should accommodate longer, easier-to-read text.

ii) Where references to command "types" is now allowed the new package should allow the full freedom of refering to individual parameters.

iii) The new package should make it easier to modify current parameter values by providing a line-editing capability.

The FORTRAN package should supply all of the following user services that are available in the present package:

i) Default values that are most reasonable and/or most convenient for a given program should be inserted automatically for each input parameter.

ii) The user should be able to ascertain the options available for each input parameter. (The "Help" service).

iii) The user should be able to review the current values of each program parameter (the "Inputs" command) and should be able to alter any or all of them. Iterative review and modification should be allowed until the user is satisfied.

iv) To spare the user from the tedium of repeatedly selecting the same options, parameters should be initialized to the values automatically saved the most recent time the same program was run by the same user from the same terminal.

Why not just same user? Why terminal?

v) Commands should be readable from a disk file as well as from a terminal (indirect command specification). That file should be a standard ASCII file that the user can create and modify through the standard system text editor.

iv) The user should be protected from typing errors. Invalid input should be rejected, the user should be notified of the error immediately, and the most recent valid parameter value should be retained.

Programming Features

To the programmer the FORTRAN Standard Command package WILL be expected to look very different from the SAIL version. Despite those differences the services provided to the programmer should be similar. In particular:

i) It should be simple to determine the current value of any parameter. The values should always be in standard internal units.

ii) It should be possible for each program that uses a Standard Command to provide appropriate default parameter values that override the ones built into the package.

iii) The package should accommodate the invention of new, special-purpose commands as required by individual programs.

iv) It should include a set of library routines that can be used to define those special-purpose commands. (For example, a routine to parse the user-supplied input.)

v) There should be "hooks" available to intercept and modify the input and output to and from specific command routines. (A slight generalization of the "Special Help" and "Special Exit" features of the SAIL package.)

Structure

The structure of a FORTRAN package will have to differ from the SAIL version. The difference should be exploited to increase programmer productivity.

i) The package should consist of subprograms rather than high-level macros. As a result, debugging a program should no longer be the terribly difficult task that it is now. (The deeply nested macros of the SAIL Standard Command package convert original program text into source code totally unfamiliar to the programmer.)

ii) Program development should be speeded up because the FORTRAN package will not have to be recompiled for each minor alteration to each program it serves. (The SAIL high-level macros produce voluminous source code that becomes an intrinsic part of the program using them. They can convert a trivial program into one with large compilation requirements.)

Part 3: DESCRIPTION OF THE EXPERIMENTAL PACKAGE

An Overview

The Experimental FORTRAN Standard Commands Package, which will be designated by the initials "XFSCP" in the rest of this report, is a collection of independently-compiled subprograms. The system linker will automatically include the required XFSCP modules into any program that needs them - provided, of course, that the linker has access to the library in which the XFSCP modules are stored.

Communication between XFSCP modules and between the package and the program it serves occurs primarily through arguments in CALL statements. The two areas of "named" COMMON are defined in a small FORTRAN source file that can be "INCLUDED" in the source for any new special-purpose command. These areas do NOT have to be part of the root segment of overlaid programs.

The XFSCP should be CALLed from a MAIN program whose sole purpose is to initialize the package; the real work to be done by programs that use the XFSCP should be carried out in subroutines. In particular, for each program there must exist a subroutine named "GO" which will be called by the XFSCP whenever the package encounters the GO command in the input stream. Any text that follows the word GO in that command will be passed intact to the GO subroutine.

This arrangement has three main advantages. First, it lets the XFSCP initiate the actual work by a CALL rather than by a RETURN statement; control is therefore retained in the package where it should be. Second, it means that there is a unique entry point to the program from the package; this is good structure in principle. Third, it lets most of the package be overlaid (when necessary) by almost all of the program being served.

The XFSCP differentiates between commands on the basis of their purpose and origin. The different types of commands are treated in radically different ways.

The first type of command consists of those that request some ACTION on the part of the XFSCP. (These are: GO, HELP, EXPLAIN, INPUTS, GETCOMMANDS, SAVE, SETDEFAULTS and the various aliases for STOP. A brief description of what each of these does in the XFSCP is given in a later section of this report.) The action-requesting commands are executed directly by the XFSCP with no opportunity for intervention by the program being served. They cannot be removed from the package even when they are not needed in a given application. They must not be redefined (by a routine using the same name as one in the XFSCP). In several ways then, action-requesting commands are "protected".

The second type of command recognized by the XFSCP consists of those that assign a value to a program parameter (or to a set of parameters). This category includes all standard commands not listed in the previous paragraph. (They have not all been implemented in the XFSCP.)

The execution of parameter-setting commands is mediated through a common "master" routine which provides a convenient point at which communication to or from specific commands can be monitored, intercepted and/or altered. In extreme cases, programs may supercede the standard XFSCP version of a command by supplying their own version.

It will be very easy to increase the repertory of "standard" parameter-setting commands but this should only happen after there is consensus on the proper definition of "standard" for each proposed addition.

The third type of command recognized by the XFSCP are those defined as part of a specific program in order to serve its unique requirements. The special commands are handled through a "master" routine which is different from the one used for standard commands. The XFSCP library contains a "stub" version of the special master routine that is linked into programs that do not require special commands.

A copy of the FORTRAN source code for the "stub" should be used as the framework of an actual special master routine whenever one is required. The stub already contains the standard interface to the rest of the XFSCP and it only needs to be edited to recognize the new commands being defined. The special master routine then merely relays information between the special commands and the rest of the XFSCP.

The XFSCP library contains a set of utility routines that should help keep to a minimum the amount of programming required for each special command. There are library routines to normalize input, to parse it, to check consistancy, etc.

Note that a special command is not limited to setting parameter values; it can do anything the programmer wants it to do. In particular, the activities of these commands can represent substantial encroachments on the GO function. A certain amount of restraint is therefore appropriate since this is closely equivalent to writing program FUNCTIONs that have "side effects", a practice generally considered undesirable.

The full text of a skeletal ("do-nothing") special command is shown in a later section of this report where we describe the test bed program used in this study.

At least conceptually, each parameter-setting or special command is a self-contained entity. The action-requesting commands that accept references to parameter-setting commands carry out their

function by "polling" the routines for each command included in the program. For example, the "HELP" command does not contain ANY text; it merely scans the list of commands being used, calling upon each in turn to supply or display its own text. (Text for a given command is not duplicated; it is defined in one, and only one, location.)

This modularity ensures that one need not recompile all of the XFSCP in order to create a special command or to modify a standard one. It reduces the space requirements of the XFSCP package to what is needed for commands used. One slight drawback to this complete modularity is the absence of a COMMON area for parameter values. Programs that use the XFSCP MUST obtain the latest value of parameters by querying the appropriate command routine. Normally this only has to be done once. The exception occurs when XFSCP services are requested from within the GO phase; the value of each potentially altered parameter must be checked after every return from the XFSCP.

Initialization

The FORTRAN Standard Commands package is initialized by a call to the subroutine SCINIT. The XFSCP expects to receive the following information:

a) The name of the program being served.

b) The version number of the program being served.

c) A succinct text containing a reminder of the program's purpose. It will be displayed for the use each time the program is started.

d) A list of names that identifies the parameter-setting and special commands used by the program.

e) Four integers to be used as logical unit numbers by the package. If the numbers are zero, the package will use LUN 5 for I/O to and from the user's terminal, LUN 7 for the file in which parameter values are to be saved and LUN 8 for indirect command files. (A better method might be to write a FORTRAN equivalent of SAIL's GETCHAN to dole out LUNs on request.)

Communication Conventions

The two areas of named COMMON used by the XFSCP contain the list of Logical Unit Numbers currently assigned, the list of parameter-setting and special commands being used by the program being served, a work space shared by standard modules and a few other miscellaneous items. Further details are documented in a later section.

The subroutine for every parameter-setting and special command is defined in terms of the same set of four formal parameters ("dummy arguments"). The first is an INTEGER numeric code that designates the service being requested from the routine. The interpretation of this code is described below. The next two arguments are CHARACTER variables through which text can be passed to and from the command routines. The last argument is a REAL variable in which a parameter value can be returned to the caller. Since FORTRAN subroutine calls are "by reference", this final argument can equally well be an array of REAL variables when appropriate.

Here is the numeric code by which the first argument designates the service requested by the caller:

1- The caller wants to have all parameters defined in this command set to their default value(s). If the second argument is not blank, the caller is simultaneously specifying what the default values should be for this command for this program. The specification is a CHARACTER string identical to the one that would be typed in the input stream to set the command parameters to the desired value(s).

2- The XFSCP has encountered the name of this command in the input stream. The routine should do whatever it is supposed to do whenever the user types this command; usually to give new values to one or more parameters. The second argument is the string that followed the command name on the input line.

3- The INPUTS command has been encountered in the input stream. The caller therefore wants (in the third argument) the text that should be displayed in response. The caller will take care of doing the actual display.

4- Caller wants the string that is to be SAVE'ed. It should be supplied as the value of the third argument. (Caller will do whatever actual I/O is required.) This and the previous function should almost always be identical.

5- Caller wants to have the HELP text output on LUNTTO.

6- Caller wants the value of one or more parameters which will be expected in the subsequent arguments. The specific values to be returned are very much command-dependent. For example, the last argument could be either the address of a scalar variable or the starting address of an array of real values. The next-to-last argument can be used to return character values. Some complex commands may require that the second argument identify which among several possible responses the called wants.

Obtaining XFSCP Services During the GO Phase

The present DEC10 Standard Commands package interacts with the program user only when the program starts or re-starts. There is no provision for interaction with the user during execution of the program's GO phase. This effectively limits all of our operational programs to a quasi-BATCH mode of operation.

The XFSCP will respond to a service request originating from the GO phase of a program. In its present form, the XFSCP then lets the user do anything that could be done during initialization (i.e. before the GO phase was entered). In some cases that much freedom might be undesirable. With a little additional programming the XFSCP could limit the commands available to a user at different stages of program execution. The GO phase would merely pass a list of enabled commands to the XFSCP as part of its calling sequence.

The EDIT Command

During the study it was suggested that users would find the Standard Command package much more convenient if they could edit parameter values instead of retyping them. The model suggested was the GYPSY program used at Groningen.

A full text editor is a major piece of software, far outside of the scope of the XFSCP. It did seem possible, however, that a ONE-LINE editor with much of the desired functionality would be feasible.

A relatively small routine was written to test this conjecture. It is system dependant because it requires "transparent" I/O (control sequences are simply passed along to the program) and suppression of automatic echoing. FORTRAN programs on the VAX have access to this type of I/O through calls to the VMS operating system.

The editor recognizes only three escape sequences - all others are simply ignored. Two of the escape sequences are used to move the terminal cursor through the line being edited; the third is used to delete a character from that line. All other characters except carriage return are inserted in the line at the cursor position as they are read from the input terminal. A carriage return terminates editing of the line currently being processed.

The editor works by calling the appropriate command routine with a request for the text that is normally displayed in response to the INPUTS command. The editor displays that text but does not end the line normally; instead, the cursor is left at the end of the line that has been displayed. The text continues to be available to the editor in a memory buffer. Whenever the editor accepts from the terminal a character or control sequence (which it interprets as a command to alter the text or to change the position of the cursor)

the buffer is updated and the latest version of the line (and cursor) is rewritten in such a way that it covers the text that was previously displayed on the terminal screen. Upon receiving a carriage control character, the editor again calls the appropriate command routine, this time with a request to accept the modified text as input to the command. The loop (get text, change it, feed it back) is repeated until a carriage return is the first character received after a line has been displayed.

Editing that creates invalid input will be rejected immediately by the command. The text supplied by the command in response to the editor's next request for INPUTS will show no change from its most recent valid content. Multi-line texts (involving ellipses) must be edited one line at a time but feedback to the command routine takes place only when the full text has been edited (i.e. the loop is closed only after receipt of the carriage return for the last line in the text).

The editor is invoked by the command EDIT. If no argument is supplied, all the command routines used by the program will be polled by the editor. The current parameter values for each command in turn can be altered or accepted as they stand. The name of a parameter-setting or special command may be used as an argument for the EDIT request, in which case only those parameters set by the specified command will be edited.

The editor will be invoked automatically whenever the user types the name of a parameter-setting or special command unaccompanied by the text normally supplied as input to such commands.

Using a Graphics Cursor as Input

It has also been suggested that the FORTRAN Standard Commands package should be able to use the position of a cursor and other values obtained from a graphics device as an alternate form of input.

The XFSCP contains nothing at all that would prevent one from writing a new command that would read and properly interpreted such input. Furthermore, there is no reason why such a command could not call upon the one-line editor to enable modification of values obtained from the alternate device through the main terminal keyboard.

At present all input to all commands is handled in one routine. The GETCOMMANDS command is implimented by a simple redefinition of the symbolic logical unit number used in that routine. (That LUN is one of the items in named COMMON.) It should be utterly trivial, one or two lines of code, to define a command that changes the input LUN to a keyboard on some other device.

Deliberate Omissions

The XFSCP has dropped the notion of command "types". The reason is that all appropriate action requests in the XFSCP have two forms; they can refer either to ALL parameter-setting or special commands in the program or to an INDIVIDUAL command. Since one can list the current parameter values, set the defaults, edit or ask for help concerning any individual command, the concept of command "type" is not really very useful any more and would be a nuisance to impliment.

For the sake of convenience, the XFSCP has dropped the ability to read multiple commands from a single line of input. This feature does not seem to be particularly valuable; in fact, several long-time users of the SAIL package were unaware of its existance. When the XFSCP encounters a line containing more than one command it generates an error message and rejects ALL commands on that line.

The PARAMETER-Setting Commands in XFSCP

Not all of the presently-used standard commands have been included in the XFSCP; only enough to conduct the tests pertinent to this project. The ones that have been done and that are more or less complete are:

ANTENNA	ANTENNAS	REFANT	REFANTS
IF	IFS	IFPAIR	IFPAIRS
SOURCE	SOURCES	CALCODES	DBNAME

The ANTENNAS command includes:

the "*" notation for "all antennas",
the "* - i, j, k, ..." notation for "all but ...",
and the default assignments that accompany the WITH option.

The DBNAME command has been coded to include all the "sticky" attributes of file names and file qualifiers. This command was a good test of the code to handle continuation lines in input, saving and editing.

The indication that the commands listed above are "more or less" complete refers to the fact that none has ever actually been used in a "real" program. In fact, the portion of the commands that returns parameter values to a calling program has been deliberately ignored (for now) because it was not needed for the purposes of the study.

The ACTION Commands in XFSCP

All of the ACTION-requesting commands of the SAIL package, one new command, and a new alias for "QUIT" have been included in the XFSCP. The definition of each command may differ slightly from the SAIL version. In alphabetical order the XFSCP ACTION-requesting commands are:

EDIT — A new command that initiates the simple one-line editor which was described previously.

EXIT, FINISH, QUIT or STOP — These are all aliases for a request to wrap things up and return to the operating system.

EXPLAIN — Requests detailed information about the program. (Basically it just writes on the terminal the content of an ASCII file.)

GETCOMMANDS — Designates a file which is to serve as a new (temporary) source of input.

GO — Indicates that parameter setting is complete (at least temporarily) and that the work of the program being served should be started (or continued).

HELP — Asks for information about one (or all) program parameters.

INPUTS — Requests a listing of current value(s) assigned to all (or to an individual) program parameter.

SAVE — Requests that all current values of program parameters be recorded on a disk file.

SETDEFAULTS — Requests that the current value of all (or a single) program parameter be set to a default value which may differ from one program to another.

Various additional ACTION-requesting commands could be added but were NOT ~~been~~ tested in the XFSCP. Among the ones suggested were:

A command to transfer a program from current interactive (terminal) control to batch control. Task shedding would be very system-dependant of course.

A command to abort a job from the XFSCP. This might be as simple as a trap for the <CNTL C> character.

Part 4: DOCUMENTATION OF THE EXPERIMENTAL PACKAGE

```
C     THIS IS SCCOMMON.DCL

C     Here is the list of recognized ACTION-type commands.
C     It is initialized by the BLOCK COMMON called SCOMMANDS.

      PARAMETER (NNSC=12)
      CHARACTER*12 SCCNAM(NNSC)

C     Here is the list of built-in PARAMETER-setting commands.
C     It is initialized by the BLOCK COMMON called SCOMMANDS.

      PARAMETER (NNSP=40)
      CHARACTER*12 SCPNAM(NNSP)

C     The following array will contain the name of all commands used
C     by a program (including the name of special-purpose commands).
C     It is initially set to blanks in the BLOCK COMMON.

      PARAMETER (NNUX=50)
      CHARACTER*12 SCPUSN(NNUX)

C     This flag tells of the current use of an auxiliary input file
C     (consequent to a GETCOMMANDS command). XXXX is a dummy.

      LOGICAL*1    USEAUX, XXXX

C     This is the name of the file used for saving parameter values.

      CHARACTER*20 FILENM

C     These are the default logical unit numbers used by the package.

      INTEGER    LUNTTI, LUNTTO, LUNCOM, LUNAUX

C     Here is a work area. It is placed in COMMON so that it can be
C     shared by many routines. User-defined routines should keep their
C     grubby little hands off of it.

      CHARACTER*400 WRKSTR

C     There are two named COMMON areas because some versions of FORTRAN 77
C     (such as on the PDP11) can't mix character variables with other
C     types in the same COMMON. (Isn't that silly !)

      COMMON /STDCO1/ FILENM, LUNTTI, LUNTTO, LUNCOM, LUNAUX,
     1                NSPUSD, USEAUX, XXXX

      COMMON /STDCO2/ SCCNAM, SCPNAM, SCPUSN, WRKSTR
```

```
      BLOCK DATA SCOMMANDS

      INCLUDE 'SCCOMMON.DCL/LIST'

C     This counts the number of parameter-setting (or user-defined)
C     commands used by the program.

      DATA NSPUSD/0/

C     This initializes the area used for selected commands

      DATA SCPUSN /NNUX*' '/

C     Here are the default LUN assignments

      DATA LUNTTI, LUNTTO, LUNCOM, LUNAUX /5,5,7,8/
      DATA USEAUX /.FALSE./


C     This is the list of Standard PARAMETER-setting commands

      DATA SCPNAM /
     1    'TIMERANGE',     'SOURCES',       'SOURCE',
     3    'DBNAME',        'CALCODES',      'CALIBRATION',  'PASSFLAG',
     3    'ANTENNAS',      'ANTENNA',       'REFANTS',      'REFANT',
     3    'IFS',           'IF',            'IFPAIRS',      'IFPAIR',
     3    'BANDS',         'BAND',          'MODES',        'MODE',
     3    'INFILES',       'INFILE',        'OUTFILES',     'OUTFILE',
     3    'DCSADDRESSES', 'DCSADDRESS',
     3    'DATATYPE',      'STOKES',
     3    'LISTOPTIONS',   'LISTOPTION',
     3    'AVERAGE',       'AMP/JY',
     3    'LSQTOLERANCE', 'GTINTERVAL',
     3    7*' '/

C     This is the list of Standard ACTION-request commands

      DATA SCCNAM /
     1    'HELP',              'EXPLAIN',
     2    'INPUTS',            'EDIT',
     3    'SAVECOMMANDS',      'GETCOMMANDS',
     4    'SETDEFAULTS',       'GO',
     5    'EXIT', 'QUIT', 'FINISH', 'STOP'/

      END
```

```
      SUBROUTINE SCINIT (HISNAM, HISVER, TEXT, USING, L1,L2,L3,L4)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     This routine establishes the initial contact between a user
C     program and the Standard Command package.
C
C     The arguments are:
C
C     HISNAM        The name of the program being served.
C     HISVER        The version number that program supplies.
C     TEXT          A message (or salutation) to the user.
C     USING         A list of commands to be used.
C     L1,...L4      Logical unit numbers for I/O. If zeroes,
C                   defaults will be supplied.
C
C     This routine never RETURNs to the caller. Upon recognizing
C     the GO command in the input stream it calls a subroutine
C     called "GO" which is expected to contain the GO phase of the
C     program being served.
C
C     The GO subroutine is expected to RETURN here so that the XFSCP
C     can resume its dialog with the user and/or exit properly at
C     the user's request.
C
C     (The SCSUB routine is provided so that XFSCP services can be
C     requested by the GO phase of a program without recursiveness.)
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) HISNAM, TEXT, USING
      REAL          HISVER
      INTEGER       L1, L2, L3, L4
```

```
      SUBROUTINE SCSUB (PROMPT)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     This routine is nearly identical to the SCINIT routine except
C     that it is called from within the GO phase of a program.
C     This lets the program interact with the user during the
C     execution of a program as well as during initialization.
C
C     This is NOT an infinite loop; it returns to the place from which
C     it was called (in the GO phase) when GO is found in the input.
C
C     The initial value of the argument is a prompting message (supplied
C     by the calling program)  which will be displayed to the user when
C     the XFSCP resumes its interaction with him or her.
C     Upon returning to the GO phase, the XFSCP will leave in this same
C     variable the string found in the GO command.
C     ------------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) PROMPT
```

```
      SUBROUTINE SCOPEN (X, NEW)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     This routine constructs the name of a file in which to SAVE the
C     parameter values for this program.
C
C     The file name reflects
C
C     1- The name of the program being served by XFSCP.
C        (Obtained from the value of the first argument.)
C
C     2- The user's identity (obtained from the operating system).
C
C     3- The point from which the program is being controlled.
C        (The terminal number or batch controller id obtained from the
C        operating system.)
C
C     This routine tries to find and open an existing file with this name.
C     If it cannot do so it creates a new one. The value of the second
C     argument is a flag that asserts wether or not a new file was
C     created. (If so, default values for all parameters will be solicited
C     from the command routines. If not, the parameter values contained
C     in the pre-existing file will be read as the initial values for
C     this run of the program.
C
C     Obviously, the name and file-handling parts of this routine are
C     system dependant.
C
C              THIS VERSION IS FOR THE VAX VMS SYSTEM
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*)  X
      LOGICAL        NEW
```

```
      SUBROUTINE SCPARS (INSTR, CMND, ITYPE, INDX)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     This is a parsing routine.
C     The first argument is a character string typed in by the user.
C     We break off the first identifiable substring (using NEEDCH)
C     and return it as the value of the second argument.
C     We search for this second argument in the user's list of defined
C     commands.
C
C     We return in the third argument a code as follows:
C
C       Neg   Command is ambiguous. (INDX is meaningless.)
C
C       Zero  Input is not a recognizable command. It may be standard
C          but not among those selected by the user program. If
C          this is the case the fourth arg (INDX) will be non-zero.
C
C         1   A PARAMETER-setting command.
C          In this case INDX will be pointing to the entry number
C          in the list of selected commands.
C
C         2   A User-defined (EXTRA) command.
C          In this case too INDX will be pointing to the entry number
C          in the list of selected commands.
C
C         3   An ACTION-request command. In this case INDX will point
C          point to the entry in the list of ACTION commands.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) INSTR, CMND
      INTEGER       ITYPE, INDX
```

```
      SUBROUTINE SCPARM (NAME, REQUES, INSTR, OUTSTR, VALUE)
C     ------------------------------------------------------
C
C     THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     This is the "master" command routine for all of the standard
C     commands that are part of the package. It fans out the call
C     for service to the proper command-defining routine.
C
C     The program being served by the XFSCP may include a modified
C     copy this routine as part of its own definition. This might be
C     desirable in two circumstances:
C
C     1- The user wants to intercept communication to or from a
C        standard command in order to impose certain restrictions
C        or alterations upon the normal command. (This routine
C        therefore provides what is known as a "hook".)
C
C     2- References to unused commands can be removed thereby paring
C        the memory requirements of the XFSCP.
C
C     The first argument is the name of the command that has been found
C     in the input stream. The others are meant to be passed to that
C     command.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      INTEGER         REQUES
      CHARACTER*(*)   NAME, INSTR, OUTSTR


      SUBROUTINE SCXTRA (NAME, REQUES, INSTR, OUTSTR, VALUE)
C     ------------------------------------------------------
C
C     THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C
C     This is a "stub".
C
C     It is the default "master" command routine used when there
C     is no need for any special-purpose commands.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) NAME,INSTR,OUTSTR
      INTEGER REQUES
```

```
      SUBROUTINE SCSAVE
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C          THIS ROUTINE DEFINES THE STANDARD COMMAND: SAVE
C
C     This routine writes out to the COMMAND file the name of the
C     parameter-setting and user-defined commands along with the current
C     value(s) of each parameter set by these commands. The file format
C     is such that it can be edited (using the standard system editor)
C     and it can be read as an alternate input source
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'
```

```
      SUBROUTINE SCGETC (INSTR)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C        THIS ROUTINE DEFINES THE STANDARD COMMAND: GETCOMMANDS
C
C     The argument is a name of the file which is to be used as the
C     new input stream. (The name must be supplied in the current
C     input stream). This routine merely OPEN's the file and sets a
C     flag; reading from this file is done by the same routine that
C     reads from the normal input stream. Input from the terminal is
C     resumed automatically when an End-of-File is encountered.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) INSTR
```

```
      SUBROUTINE SCAMEN
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     THIS ROUTINE DEFINES THE STANDARD COMMANDS: EXIT, QUIT, AND STOP
C
C     This is the wrap-up routine. It calls a "stub" with the
C     name SCXIT so that the user can write his own version
C     (if needed) to take appropriate concluding actions such as
C     closing files.
C
C     Each command used by the program is solicited for a character
C     string to be recorded in the SAVE file. Presumably this will
C     reflect the latest value of each parameter.
C
C     Then it's bye-bye.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'
```

```
      SUBROUTINE SCXIT
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     This is a "stub".
C
C     It is the default called by the AMEN routine at wrap-up time.
C     This technique lets the user write his own version (which will
C     automatically supercede this one) to take any final action
C     that might be appropriate (such as closing files).
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'
```

```
      SUBROUTINE SCEDIT (INSTR)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C          THIS ROUTINE DEFINES THE STANDARD COMMAND: EDIT
C
C
C     This routine edits the line printed for parameter values
C     It uses the secondary routine SCEDX for system-dependent
C     operations.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) INSTR
```

```
      SUBROUTINE SCEDX (WRK, NCH, ED)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     It contains the system-dependent portion of the code for the
C     line editor. In particular, this is the portion that reads
C     one character at a time (in "transparent" and "no-echo" mode)
C     from the input terminal. It echoes back the line being edited
C     so that character deletion and insertion as well as curser
C     motion relative to the text appear correct to the user.
C     ------------------------------------------------------------

      INCLUDE '($IODEF)'

      LOGICAL*1      ED
      INTEGER        NCH
      CHARACTER*(*) WRK
```

```
      SUBROUTINE SCGTIN (INSTR)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     Reads the input from a terminal or from an auxiliary input file.
C     It squeezes out redundant blanks, checks for multi-line input
C     (designated by the ellipsis ...) and concatenates that input
C     into one long character variable. The result is NOT parsed
C     here; it is merely passed along to the caller.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) INSTR
```

```
      SUBROUTINE SCINPU (INSTR)
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C         THIS ROUTINE DEFINES THE STANDARD COMMAND: INPUTS
C
C     If the argument is blank this routine will poll all of the
C     command routines used by a program to obtain from each the
C     current value of the parameters (which this routine displays
C     in one or more lines as appropriate.)
C
C     If the argument is not blank it is expected to be the name of
C     a single command whose current parameter value(s) are to be
C     displayed.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      CHARACTER*(*) INSTR
```

```
      SUBROUTINE SCRDIN
C     ------------------------------------------------------------
C
C      THIS IS AN INTEGRAL PART OF THE STANDARD COMMANDS PACKAGE
C
C     This routine is triggered whenever the initialization routine
C     finds that there exists a file with parameter values saved by
C     a previous run of the program. This routine will read the file,
C     call the parser, relay the input to the appropriate "master"
C     command routine and go back for more until an End=of-File is read.
C
C     Notice that this routine is not set up to handle ACTION-requesting
C     commands which are not stored in the SAVE file.
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'
```

Part 5: UTILITY ROUTINES

SUBROUTINE MNMTCH (IN, LIST, ISTAT)

```
C     ------------------------------------------------------------
C     This routine does a MINMATCH of the input CHARACTER variable
C     with a list of similar variables concatenated (with suitable
C     delimiters) into one long character variable.
C
C     If a unique match is found the value of ISTAT will be the
C     sequence number of that entry in the list.
C     If no match is found ISTAT will be zero.
C     If ambiguous, ISTAT will be negative.
C     ------------------------------------------------------------

      CHARACTER*(*) IN, LIST
```

SUBROUTINE LSMTCH (IN, TABLE, NOPTS, ISTAT)

```
C     ------------------------------------------------------------
C     This routine does a MINMATCH of the input CHARACTER variable
C     with a table of similar variables. In other words, it does the
C     same sort of thing that MNMTCH does but it operates against an
C     array of character variables (the number of array elements
C     is NOPTS) rather than against one long character variable.
C
C     If a unique match is found the value of ISTAT will be the
C     index of that entry in the table.
C     If no match is found ISTAT will be zero.
C     If ambiguous, ISTAT will be negative.
C     ------------------------------------------------------------

      CHARACTER*(*) IN, TABLE(1)
```

```
      SUBROUTINE NEEDCH(IN,OUT,ISTAT)

C     ------------------------------------------------------------
C     This routine picks up the first character string from an
C     input line. The string is terminated by the first blank, tab,
C     comma or semicolon (or the end of input) unless the string
C     starts with a quote mark.
C
C     Leading blanks are removed from the input. If the string
C     is longer than the length of the CHARACTER variable sent to
C     fetch it the extra characters go in the "bit bucket".

C     The value of ISTAT returned is the length of the output
C     string (up to, but not including, the terminating character).
C     ------------------------------------------------------------
```

```
      SUBROUTINE WHATUN (INSTR, CLASS)

C     ------------------------------------------------------------------
C     This routine will examine the input string (INSTR) to see if the
C     first field is a numeric value. If it is not, the value of CLASS
C     will be set to '*'. If the input string does start with a number
C     the class of the units field will be returned in CLASS.
C     ------------------------------------------------------------------

      CHARACTER*(*)  INSTR, CLASS
```

```
      LOGICAL FUNCTION ISNM (INSTR, CLASS)

C     ------------------------------------------------------------
C     This routine will examine the input string (INSTR) to see if the
C     first field is a numeric value.
C     The units of the value will be compared with the specified CLASS.
C     If there is a number and the units are of the appropriate class
C     then the value returned will be TRUE otherwise FALSE.
C     ------------------------------------------------------------

      CHARACTER*(*)  INSTR, CLASS
```

```
      SUBROUTINE NEEDNM (INSTR, CLASS, VALUE, ISTAT)

C     ------------------------------------------------------------
C     This routine will pick up the first numeric value from an input
C     string (INSTR). The value will be returned in the second argument.
C     Note that it will be returned as a REAL variable.
C     The units of the value will be compared with the specified CLASS.
C     If there is a number and the units are of the appropriate class
C     then the value returned will be scaled into the proper internal
C     units. If all goes well the value of ISTAT will be returned as 1.
C     Other values of ISTAT are coded as follows:
C
C         0    Input string did not start with a numeric value
C        -1    Units of input did not match units requested.
C
C     If, and only if, all went well (ISTAT=1) the numeric field
C     and the units symbol will both be stripped off the front of the
C     input string. The remainder of the string will be left normalized.
C     ------------------------------------------------------------

      CHARACTER*(*)  INSTR, CLASS
```

```
      LOGICAL FUNCTION ISFL (IN)

C     ------------------------------------------------------------------
C     This routine tests the first substring of an input line to determine
C     if it contains a file designator. The string is terminated by the
C     first blank, tab, comma or semicolon (or the end of input).
C     The string must NOT be enclosed within quote marks.
C
C     CLEARLY, THIS ROUTINE IS OPERATING-SYSTEM DEPENDANT. THIS VERSION
C              IS FOR THE VAX SYSTEM VMS.
C
C     ------------------------------------------------------------------

      CHARACTER*(*) IN
```

```
      SUBROUTINE NEEDFL(IN,OUT,NODE,DEV,DIR,FILE,EXT,VER,DISP,ISTAT)

C     ------------------------------------------------------------------
C     This routine tries to obtain a file name from the first substring
C     of an input line. The string is terminated by the first blank, tab,
C     comma or semicolon (or the end of input). The string must NOT be
C     enclosed within quote marks.
C
C     CLEARLY, THIS ROUTINE IS OPERATING-SYSTEM DEPENDANT. THIS VERSION
C              IS FOR THE VAX SYSTEM VMS.
C
C     The full file name is returned in the string OUT where various
C     default values will have been supplied by the routine.

C     Default file name components are expected in the subsequent arguments
C     which will be returned with the components actually used.

C     If a disposition field is found in the input string it will be
C     returned in the next-to-last argument (DISP).

C     The last argument (ISTAT) will be set to 1 if all goes well,
C     or to 0 otherwise.
C
C     ------------------------------------------------------------------

      CHARACTER*(*) IN, OUT, NODE, DEV, DIR, FILE, EXT, VER, DISP
```

```
      LOGICAL FUNCTION ISLETR (ARG)

C     ------------------------------------------------------------
C     This function is .TRUE. if the first (or only) character in the
C     input string is a letter (upper or lower case).
C     Otherwise it is      .FALSE.
C     ------------------------------------------------------------

      CHARACTER*(*) ARG


      LOGICAL FUNCTION ISDIGT (ARG)

C     ------------------------------------------------------------
C     This function is .TRUE. if the first (or only) character in the
C     input string is a digit (0-9).
C     Otherwise it is      .FALSE.
C     ------------------------------------------------------------

      CHARACTER*(*) ARG


      LOGICAL FUNCTION ISDLIM (ARG)

C     ------------------------------------------------------------------
C     This function is .TRUE. if the first (or only) character in the
C     input string is a delimiter (blank or tab or comma or end of line).
C     Otherwise it is      .FALSE.
C     ------------------------------------------------------------------

      CHARACTER*(*) ARG


      SUBROUTINE FNDLIM (IN, WHERE)

C     ------------------------------------------------------------------
C     This routine is used to find the first occurrance of a delimiter
C     (blank or tab or comma or end of line) in the input string (IN).
C     ------------------------------------------------------------------

      CHARACTER*(*) IN
      INTEGER       WHERE
```

```
      SUBROUTINE FNDLAS (IN, WHERE)

C     ------------------------------------------------------------------
C     This routine is used to find the last occurrance of a non-blank
C     character in the input string (IN).
C     ------------------------------------------------------------------

      CHARACTER*(*) IN
      INTEGER       WHERE


      SUBROUTINE SQUEEZ (IN, OUT, LAST)

C     ------------------------------------------------------------------
C     This routine removes redundant blanks from an input string.
C     (A redundant blank is one that follows another blank or a comma.)
C     It is careful to leave quoted strings untouched.
C     ------------------------------------------------------------------

      CHARACTER*(*) IN, OUT


      SUBROUTINE LJUST(IN)

C     ------------------------------------------------------------------
C     This routine strips leading blanks and tabs from an input string
C     i.e. Left Justify it.
C     ------------------------------------------------------------------

      CHARACTER*(*) IN


      SUBROUTINE UPPER (IN, OUT)

C     ------------------------------------------------------------------
C     This routine translates all lower case letters to upper case
C     ------------------------------------------------------------------

      CHARACTER*(*) IN, OUT
```

Part 6: THE "TEST BED" PROGRAM

Introduction

The purpose of this program was to test various features of the experimental FORTRAN Standard Commands package. At this time it should also serve as an illustration of the way in which various features of the package can or should be used.

The MAIN program

The MAIN program has been kept VERY small; it serves no other purpose than to initialize the XFSCP. As explained above, all of the work to be performed by the program served by the XFSCP should be done in subroutines.

PROGRAM TSTBED

```
C     ------------------------------------------------------------
C     This program is not meant to do anything usefull. It is merely
C     a "test bed" for experimentation on a FORTRAN version of the
C     standard commands package.
C
C     The MAIN program merely calls the package to initialize it.
C     The package will NEVER return. It calls GO to do the work.
C
C     The list of commands is in 3 parts only for aesthetic reasons.
C     ------------------------------------------------------------

      CHARACTER*80   TEXT
      DATA TEXT /'  I test the EXPERIMENTAL Standard Command Package'/

      REAL*4         VERSN
      DATA  VERSN    /1.1/

      CHARACTER*40   USE1, USE2, USE3
      DATA USE1     /'IF, IFPAIRS, REFANTS, SOURCES,'/
      DATA USE2     /'CALCODES, ANTENNAS, DBNAME,'/
      DATA USE3     /'MY.EXTRA.ONE, MY.SECOND'/

      CALL SCINIT('TSTBED',VERSN,TEXT,USE1//USE2//USE3,0,0,0,0)

      STOP
      END
```

The GO subroutine

This routine is the entry point for all action performed by the program being served by the XFSCP. It is called whenever the package receives the command "GO". Any character string that follows the word "GO" is passed to this subroutine as an argument. (The string is not parsed in any way but consecutive blanks may have been squeezed down to single blanks.)

```
      SUBROUTINE GO (TYPE)

C     ------------------------------------------------------------
C     This will be the GO phase. In general, the GO phase is the
C     main part of the application program. The argument is the
C     character string (if any) that follows the word GO in the
C     command. This allows for many different action requests.
C     ------------------------------------------------------------

      CHARACTER*(*)   TYPE

      CHARACTER*(60)  TEXT, GOSTR
      DATA TEXT  /'This string is sent to the user via the XFSCP'/

      I=LEN(TYPE)
      IF (I .GT. 70) I=70

      WRITE (5,100) TYPE(1:I)
100   FORMAT('0 On initial entry to the GO phase',
     1         ' the string received was:'/'  ',A)

      GOSTR=TEXT
      CALL SCSUB(GOSTR)


      WRITE (5,200) GOSTR
200   FORMAT('0 Back from the Command package which returned',
     1    ' to the GO phase with:'/'  ',A)

      RETURN
      END
```

The SCXTRA subroutine

This is an example of a "master" routine for user-defined commands. It is called by the standard package whenever any user-defined command is recognized in the input stream. This routine need NOT be supplied in cases where no special commands have been defined. For such cases a dummy routine of the same name (a "stub") has been included in the Standard Commands library.

Since the first argument is the name of the command that has been encountered, only one master routine is required no matter how many special commands have been defined. (Note that the name being passed will always be fully expanded and in upper case.) Funneling all references to user-defined commands through one conventionally-named "master" routine is the only way one default routine can satisfy linkage requirements.

All but the first argument are meant to be passed along to the command-defining routine. They are described elsewhere. Of course the programmer always has the option of combining the command master and the code for the actual commands. This would be especially appropriate if only one special command is being defined.

```
      SUBROUTINE SCXTRA (NAME, REQUES, INSTR, OUTSTR, VALUE)

C     ------------------------------------------------------------
C     This is a sample of a user-defined command master.
C
C     The first argument is the name of a command. The others are
C     to be passed along to the routine for that specific command.
C     ------------------------------------------------------------

      INCLUDE  'SCCOMMON.DCL/NOLIST'

      INTEGER        REQUES
      CHARACTER*(*) NAME, INSTR, OUTSTR
      REAL           VALUE

      IF (NAME .EQ. 'MY.EXTRA.ONE') THEN
          CALL MYOWN1 (REQUES, INSTR, OUTSTR, VALUE)

      ELSE IF (NAME .EQ. 'MY.SECOND') THEN
          CALL MYOWN2 (REQUES, INSTR, OUTSTR, VALUE)

      ELSE
          WRITE (LUNTTO,100) NAME
100       FORMAT('  I don''t recall defining something called ',A)
      END IF
      RETURN
      END
```

Special purpose (user-defined) commands

The following page shows an example of a user-defined command; It doesn''t do a whole lot. A second special command was written but it is just a copy of the first and is not shown.

The argument list in this example is standard but, of course, the interface to the XFSCP is in the master routine (SCXTRA - described just previously) so the user is free to devise whatever argument list is appropriate.

```
      SUBROUTINE MYOWN1 (REQUES, INSTR, OUTSTR, VALUE)
C     ------------------------------------------------------------
C     This is a sample of a user-defined command.
C
C     Its single parameter is a character string which will contain
C     a message describing what the command has done most recently.
C     (The string becomes whatever is fed to the command in the input.)
C     ------------------------------------------------------------

      INCLUDE 'SCCOMMON.DCL/NOLIST'

      INTEGER        REQUES
      CHARACTER*(*) INSTR, OUTSTR
      REAL        VALUE
      CHARACTER*60  MYVAL

      IF (REQUES .EQ. 1) THEN
          MYVAL='MY first command has been set to its DEFAULT value'

      ELSE IF (REQUES .EQ. 2) THEN
          MYVAL=INSTR

      ELSE IF ((REQUES .EQ. 3) .OR. (REQUES .EQ. 4)) THEN
          OUTSTR=MYVAL

      ELSE IF (REQUES .EQ. 5) THEN
          WRITE (LUNTTO,200)
200       FORMAT('0 MY.OWN.EXTRA',T20,'This is the HELP text')

      ELSE IF (REQUES .EQ. 6) THEN
          OUTSTR=MYVAL

      END IF
      RETURN
      END
```

Part 7: ESTIMATES OF TIME AND EFFORT

Completion of the VAX package

Since the design work has been done and the fundamental service routines have been written, the remaining work is routine coding and debugging. It should take between 2 and 3 weeks.

Clearly it would take longer if the final specifications for the package entail major changes from what has already been done in the XFSCP.

A PDP-11 Version

The problems of writing the package with the PDP11 FORTRAN 77 subset will make this a rather painful chore.

A reasonable estimate of the amount of time required is about 2 months.

Re-writing Pipeline Software to Include the Package

Once the package is available on the PDP11 (cf above) it should not take more than 1 to 2 weeks to revise both the DBUTIL and DBFILLER programs to make use of it.

A DEC10 Version

Since DEC has announced plans to deliver a full implementation of FORTRAN 77 for the DEC10 in the first quarter of next year (1983), we should postpone any work on a FORTRAN Standards Command package until at least that time. Alternatively, we could start as soon as we wanted to by using the DEC20 version of FORTRAN 77 which is already available at various locations among them New Mexico Tech in Socorro. (The DEC20 version of FORTRAN 77 and the DEC10 version should be identical.)

When FORTRAN 77 is available on the DEC10, converting the VAX version to work on the DEC10 will involve no more than rewriting the system-dependant routines. That should take no more than a week to 10 days.

Rewriting DEC10 Application Programs to use the Package

This is a major undertaking and, IF IT IS TO BE DONE AT ALL, it should be planned as an ongoing project whereby each component of the present SAIL library is rewritten at the time major maintenance would otherwise be done on that program. (For example, when modifications are needed for Spectral-line work or to increase the number of continuum channels to 4.)

A reasonable guess for the total effort required to convert all of our present programs is 18 +/- 3 man-months.

It is NOT the intent of this report to recommend one way or the other about the desirability of embarking on such a conversion project. At the very least a decision on this matter should await the completion of the larger language study that is planned.