300-FOOT CONTROL COMPUTER MEMO NO. 7

PROPOSED CONTROL LANGUAGE

FOR THE 300 FOOT SYSTEM

October 31, 1984

Allen Farris
N. R. A. O.
Computer Division
Green Bank

INTRODUCTION

The control language for the new 300 foot system will attempt to draw on what the user already knows by using statements common to familiar computer languages. It will be an interactive language, in the sense that a user will have the ability to stop execution at any point and display or change statements that have been submitted to the control program. It is intended to support a novice user, without creating undue trauma, as well as the user who wants extensive control over the system. This goal will be achieved by building defaults into the language and giving the user control over the assignment of these default values. This technique will form the basis for constructing an optional menu-driven system which will guide the user through the setup and observing procedures. In this manner, the language will be used to construct a system in which a knowledge of the language is unnecessary.

A control language, for the purposes of this discussion, is a coherent, consistent, and unambiguous set of instructions for directing the control program. There are two levels at which the control system operates. At the lower level the control program activates, controls, and interacts with various hardware devices related to timekeeping, telescope motion, and data acquisition. At the higher level the control program must interact with the user and execute instructions more from the user's perspective, e.g. "observe this object for this length of time." These higher level instructions divide into two groups: 1) instructions designed to establish the proper environment in which to carry out some subsequent command, e.g. setting an LO frequency, and 2) instructions designed to directly initiate some major activity, e.g. start a scan. This document is concerned with this higher level set of instructions.

The difficulties of defining such a language become apparent when one begins to reflect on the community of users of such a language. This community includes astronomers with little familiarity with the telescope, astronomers with greater familiarity who want somewhat more control over the observing situation, and experienced astronomers who want control over everything and do nothing in a conventional manner. The community also includes telescope operaters who are very familiar with the instrument and have day-to-day contact with the system, engineers who must install and check out new equipment, engineers and programmers called out to diagnose and solve a problem during an observing session, and programmers and engineers assisting an astronomer to observe using an unusual technique. The needs of such a community of users diverge in the extreme. At one end of the spectrum a turn-key operation is desirable. At the other, information about and control over every facet of the system is a necessity. This is the major problem in considering a command language.

It is highly desirable to be able to submit instructions to the control program on a dynamic basis and be able to interrupt and

modify them. One must also keep in mind that these instructions are being executed in real-time, so efficiency is of paramount importance.

These considerations are summarized in the following series of desirable goals for the command language. The command language must:

1. establish and initialize an environment for executing major activities
2. execute major activities
3. operate with real-time efficiency
4. be interruptible, with dynamically submitted instructions
5. be easy to use for inexperienced users
6. provide complete run-time control for sophisticated users
7. provide flexibility and generality for ease of maintenance and growth

CONTROL FLOW STATEMENTS

The control flow statements of the language are common to many high level computer languages and require little explanation. They are presented below.

1.
```
WHILE (conditional_expression)
    ...
ENDWHILE
```

2.
```
IF (conditional_expression)
    ...
ELSEIF (conditional_expression)
    ...
ELSEIF (conditional_expression)
    ...
ELSE
    ...
ENDIF
```

3.
```
REPEAT
    ...
UNTIL (conditional_expression)
```

4.
```
DO loop_index = beginning_value, ending_value, increment
    ...
ENDDO
```

5.
```
EXIT
```

Looping statements and 'IF' statements may be nested to some arbitrary maximum. The 'elseif' and 'else' clauses are optional. The 'DO' statement is entirely equivalent to:

```
loop_index = beginning_value
WHILE (loop_index <= ending_value)
        ...
        loop_index = loop_index + increment
ENDWHILE
```

The 'EXIT' statement jumps to the statement following the end of the loop in which it occurs.

STRUCTURAL STATEMENTS

The language, from a structural perspective, consists of one or more procedures, similar to Fortran subroutines, and one or more groups of statements called observations. It is observations that are executed by the control program. Of course, observations may reference procedures. The statements which form these groups are presented below.

1.
```
OBS user_number observer_name project_code
    ...
ENDOBS
```

2.
```
PROC procedure_name
    ...
ENDPROC
```

Procedure groups and observation groups may not be nested, i. e. an observation group cannot contain within its boundaries the definition of a procedure and likewise a procedure group cannot contain within its boundaries the definition of an observation or of another procedure. Observation groups may contain references to procedures but procedures may not contain references to observations. In other words observation groups are similar to Fortran 'main' programs and procedure groups are similar to Fortran subroutines. Procedure groups may contain references to other procedures. Procedure and observation groups may be in any order. Observation groups are executed by the control program in the order in which they occur. There are no formal parameters because all variables are globally defined, including user-defined variables. These two groups of statements may contain any of the control statements, data declaration statements, assignment statements, or procedure references.

DATA DECLARATION, ASSIGNMENT, AND PROCEDURE REFERENCE STATEMENTS

Additional types of statements include data declarations, assignments, and procedure references. These are presented below.

1. CHAR variable_name
2. INT variable_name [(dimensions)]
3. REAL variable_name [(dimensions)]
4. DOUBLE variable_name [(dimensions)]
5. variable = expression
6. procedure_name

Types of variables are character, integer, real, and double precision. Character data is in the form of strings of varying length. The only context in which character string constants may occur is in character assignment statements. Character variables may not appear in expressions. Integer, real, and double precision variables may optionally have dimensions and thus become arrays and are referenced as in Fortran. Expressions are formed as in Fortran and may contain function references. Such functions are all built-in, such as SIN, COS, etc. Procedure references may not occur in expressions. Procedures are referenced and executed merely by stating their names.

DEFAULTS

The ability to define and manipulate defaults is an important concept, which, if properly executed, contributes significantly to the ease with which the language can be used. The statements of this type will most frequently be used to pre-define names and defaults which the user may then manipulate.

Defaults are handled in the following manner. Elementary variables may be grouped into logically related collections and referred to by a single name. Likewise, collections of constants which might be assigned to these variables may be given a name. There may be many groups of constants which might be assigned to a single collection of variables. The assignment of constants to variables is made in the usual manner: name of a collection of variables = name of a collection of defaults. This feature of the language will ordinarily be used to pre-define meaningful collections of variables as well as commonly used default values for these variables.

The statements used to define collections of variables, collections of default values for these variables, and the assignment of those values to those collections are given below.

1. SET set_name
       set_membership_specification
   ENDSET

2. DEF default_name OF set_name
       default_item, default_item, default_item,...;
       default_item, default_item, default_item,...;
       default_item, default_item, default_item,...;
       ...
   ENDDEF

3. set_name = NULL
   set_name = default_name

A set_membership_specification may be as simple as a list of variable names separated by commas. A default_item in the default definition may be of the form:

        variable_name = constant
or,   system_defined_default_name
or,   constant

The default definition merely serves to give a name to a collection of assignment statements which may be invoked in assigning values to a set of variables. The ';' in the default definition serves to delimit groups of assignment statements. The items in the list separated by ';' form a specification of values for the named set and are referred to in a manner similar to elements of an array, viz. default_name(1), default_name(2), etc.

The assignment statement of the form 'set_name = NULL' initializes all variables of that set to a null value, indicating that the variable has no value. All variables retain their values until explicitly changed by an assignment statement.

An example will make these concepts clearer. Consider a list of sources. There will be a system defined set called SOURCE, which, among other variables, will contain SOURCE_NAME, HORIZONTAL_POSITION, VERTICAL_POSITION. The system defined SOURCE_LIST is merely a list of sources. A default definition might take the following form:

```
DEF S OF SOURCE_LIST
        source_name_1, horizontal_position_1, vertical_position_1;
        source_name_2, horizontal_position_2, vertical_position_2;
        source_name_3, horizontal_position_3, vertical_position_3;
        source_name_4, horizontal_position_4, vertical_position_4;
        ...
ENDDEF
```

The items in the source list may be assigned, as in:

```
SOURCE = S(2)                    , or

SOURCE_LIST = S                  ,
```

which assigns the list of sources to the system variable SOURCE_LIST, which may then be used by some defined procedure to begin data collection. In the special case of sources, an individual specification of values pertaining to a particular source may be referred to by its source_name. In other words, the values

source_name_i, horizontal_position_i, vertical_position_i;

may either be referred to as S(i) or source_name_i, whichever suits best in the particular context. 'SOURCE = source_name_i' might be desirable in many contexts, while 'SOURCE = S(i)' might be desirable within execution loops.

The complete specification of system defined sets and default values for those sets will form an important aspect of the system for the user community. A user will also have the option of specifying collections of statements, indexed by project-id and stored on the system disk, which will automatically be executed at the beginning of an observation.

ADDITIONAL STATEMENTS, ESPECIALLY OF AN INTERACTIVE NATURE

Especially for problem diagnosis greater control is needed over the execution of statements in real-time, and also the ability to stop execution and display the status of variables. The statements of the command language designed to accomplish this are displayed below.

1. PAUSE [AFTER source_code_line_no]

2. RESUME [WHEN (conditional_expression)] [AT source_code_line_no]

3. SHOW STATUS
   SHOW character_string_constant_or_character_variable
   SHOW list_of_variables_or_sets

These statements may be submitted in a "batch" mode like the other statements in the language or interactively. 'SHOW STATUS' will display the statements currently submitted to the control program with line numbers and also display where the control program is currently executing. 'SHOW character_string...' merely displays the character string, while specifying a list of variables or sets will display the current values of those variables. The clauses in square brackets are optional and provide additional control capabilities.

The following statements may be entered interactively. The first two may be used to alter statements already submitted to the control program. These statements may be used to change user-submitted statements only. They may not be used to alter system defined procedures, system defined sets, or system defined defaults.

4. DELETE source_code_line_no THRU source_code_line_no

5. INSERT AFTER source_code_line_no
   ...
   ENDINSERT

6. ?
   key_word?
   ??

The insert group of statements may contain statements of any type and likewise the delete statement may delete any statement. The '?' statement gives a brief syntax summary of commands in the language. The 'key_word?' form gives a syntax summary of the command associated with that keyword, and the '??' form gives a one line identification of system defined sets and defaults. Assignment statements may also be submitted interactively and used to alter the values of variables during execution.

A MENU-DRIVEN APPROACH

For many purposes it appears that a command language of some power is required. However, for simple and straightforward tasks it appears to be cumbersome. In addition it burdens the user with yet another language to learn and remember, however simple and easy to use it may appear.

One possible way to deal with this dilemma is to construct an optional menu-driven approach. The language, as outlined here, can be used to design such a system. By using the mechanism for handling defaults and by carefully defining system sets in a meaningful hierarchy, the entire user interface can be menu-driven. Each system defined procedure would have a list of required variables. If any variable failed to have a value, the user would be prompted for that value. The construction and arrangement of defaults is crucial to this approach.

In the final analysis both a powerful command language and a simplified menu-driven approach will be needed.