Command Line Arguments & Environment Variables
Everything-at-Once Model

by Craig T. Dedo
February 25, 1997

This paper is the second draft of the proposal to add command line arguments and environment variables to Fortran 2000. This revision includes the following changes:

- The use of blanks to separate command line arguments has been dropped in favor of using command argument delimiters defined by the host operating system. In case the host operating system does not define command argument delimiters, the definition of such delimiters is processor dependent.

- The language regarding the source of command arguments has been generalized so as not to state or imply that such arguments must physically follow the program name. In addition, explanatory text has been added so as to clarify that this feature is applicable to the conventions which are used with certain Graphical User Interfaces (GUIs).

- Several notes have been added to explain why features were defined in a certain way, the practical consequences of how certain features are defined, and to answer questions raised by the general public.

- The subroutine GET_COMMAND_LINE has another argument, IARGUMENT_LENGTHS, which reports the length of each command line argument. This allows for the subroutine to report the usage of significant trailing blanks on those operating systems which support such a feature.

- The subroutine GET_ENVIRONMENT has two additional arguments: INAME_LENGTH for specifying the length of the NAME argument and IVALUE_LENGTH for specifying the length of the VALUE argument.

- There are additions to the glossary for the terms "command line", "command line tail", and "operating system".

1. Rationale

Getting command line arguments and environmental variables is obviously desirable since it has been implemented in a vendor-specific manner on a wide variety of machines and a wide variety of compilers. This capability is already part of the standard functionality for C and C++.

This feature allows application developers a way of passing information to their program right at startup. Such information can be very valuable in configuring the program before passing control over to users or in performing other important tasks such as opening documents and files. This requirement requires direct access to the operating system. Right now, these capabilities are beyond the current definition of the Fortran language.

Similarly, providing a means for an executing program to obtain the program's startup command will allow the program to use this information in order to find related files that it needs or for other purposes.

The basic functionality for command line arguments is common among implementations. However, there is no de-facto standard means to specify it. Section 4 of this paper contains a list of some vendor-specific implementations.

Some systems also offer environmental variables, process-defined symbols, process-defined logical names, or system-defined logical names. Some of this functionality could be incorporated by an intrinsic which returns a processor-defined result when passed a character variable.

Not all environments have a command line to return. However, an implementation should return a status field and one status can be that there is no processor-defined command line to return. By analogy, the DATE_AND_TIME intrinsic is provided even though some systems do not have real-time clocks.

Although it is highly likely that windowing operating systems will dominate the computers of the future, it is unlikely that the need for this feature will go away. Most windowing operating systems already have a means of providing command line arguments and/or environmental variables at program startup.

It is the intent of this paper to provide the same functionality which is currently available in C and C++ compilers. Desirable features include:

- A user interface which is easily understood, easy to use, and which has the look-and-feel of Fortran.
- A user interface which allows the programmer to exercise a high degree of control. The programmer can select those capabilities which are necessary or desirable at a given point in a program while being able to ignore those capabilities which are not needed.
- Parallel syntax between the command line and environment variables.
- Ability to treat the command line as either a string or as a series of arguments. This is similar to the DATE/TIME vs VALUES option in the DATE_AND_TIME subroutine.
- Automatic parsing of the command line based on operating system defined or processor defined delimiters. Many users want this feature without the bother of doing the grunt work themselves.
- Allowing for systems which can only return the command line tail vs systems which can return the full command line.
- Ability to provide an error status including the fact that no command line can be provided or the specified environmental variable does not exist.
- Ability to provide this functionality in a variety of operating environments in as portable a way as possible using the same syntax. This includes differences in operating systems, command argument delimiters, and graphical user interfaces (GUIs).

2. Technical Specification

This proposal adds two separate intrinsic subroutines with parallel syntax. This proposal assumes that the Allocatable Extensions TR will become part of Fortran 2000 in substantially its current form.

A program's command line includes information which is associated with the program at startup, according to the rules and conventions of the program's operating system and environment. This information includes, but is not limited to, character strings included with the program's startup command.

The subroutine GET_COMMAND_LINE obtains information from the program's command line. This information includes:

- The number of command line arguments,
- The file name of the program which is executing,
- The command line tail, i.e. everything in the command line except for the program name,
- Automatic parsing of the command line tail into individual command line arguments,
- The length of each command line argument, and
- Return of the subroutine's completion status.

The subroutine GET_ENVIRONMENT obtains information from the program's environment.

Most of the arguments of these two subroutines are of type CHARACTER. Any mismatch between the length of the argument and its associated value is resolved according to the usual rules for CHARACTER assignment, i.e., the CHARACTER value is truncated or blank filled on the right as necessary. If the KIND type or character set of the target variable cannot represent the assigned value, an error occurs.

GET_COMMAND_LINE (NARGUMENTS, STARTUP_COMMAND, COMMAND_LINE, COMMAND_ARGUMENTS, IARGUMENT_LENGTHS, ISTATUS) obtains command line arguments and related information. Command line arguments appear with the command which starts the program and are separated from each other by delimiters. The processor shall use the operating system definition of command argument delimiters if such a definition is available. If the operating system definition is not available, the definition of command argument delimiters is processor dependent.

> Command argument delimiters are defined in this way in order to allow the processor the maximum amount of flexibility to implement delimiters as may be required by the host operating system. There also is flexibility for the processor to define delimiters in case the operating system does not do so.

All six arguments are optional and are INTENT (OUT).

NARGUMENTS is a scalar of type INTEGER of any kind that is defined on the processor. It is assigned the number of command line arguments which are supplied to the program. If there is no command line or command line arguments, it is assigned the value zero (0).

STARTUP_COMMAND is a scalar of type CHARACTER of any kind that is defined on the processor. It is assigned the file name of the program which is executing. The value of STARTUP_COMMAND is the same value which would be returned by executing an INQUIRE (FILE=file-name-expr) statement on the program's file name.

COMMAND_LINE is a scalar of type CHARACTER of any kind that is defined on the processor. It is assigned the value of the command line tail, i.e. everything in the command line except for the program name. If there is no command line or command line tail, it is assigned all blanks.

The command line tail need not physically follow the program name. It could come before the program name, both before and after, or be associated with the program name in some other fashion. An example of this last option would be a list of resources and other objects associated with the program at startup in a Graphical User Interface (GUI) operating system.

COMMAND_ARGUMENTS is a rank one allocatable array of type CHARACTER of any kind which is defined on the processor. Each array element is assigned the respective argument of the command line, in the order in which the argument appears on the command line. If the operating system or context does define a specific order, the order for the command line arguments is processor dependent. Each command line argument is separated from the other arguments by delimiters. If COMMAND_ARGUMENTS is not allocated at the time of a call to GET_COMMAND_LINE, it is allocated with a dimension of NARGUMENTS, and each element has the length of the longest argument. If there is no command line or command line tail and COMMAND_ARGUMENTS is not already allocated, COMMAND_ARGUMENTS has a dimension of one and is assigned all blanks.

On some operating systems, it is possible that the parsed command line arguments are available but the entire unparsed command line is not available. In such situations, it would be standard conforming for a processor to return the command line arguments in COMMAND_ARGUMENTS and, at the same time, return all blanks in COMMAND_LINE. It also would be standard conforming for a processor in such situations to return the concatenation of all of the COMMAND_ARGUMENTS values as the return value of COMMAND_LINE.

IARGUMENT_LENGTHS is a rank one allocatable array of type INTEGER of any kind which is defined on the processor. Each array element is assigned the length of the respective command line argument in the same element position in COMMAND_ARGUMENTS. Whether the length reported in an element of IARGUMENT_LENGTHS is longer than the last non-blank character of the corresponding element of COMMAND_ARGUMENTS is processor dependent.

The length of some command arguments could be longer than the position of the last non-blank character if some or all of the trailing blanks are significant.

ISTATUS is a scalar of type INTEGER of any kind that is defined on the processor. On return from the subroutine, ISTATUS is assigned the operating system completion status if it is available. Otherwise, the value of ISTATUS is processor dependent.

Example
Assume that the operating system's status code for "success" is 1, the executing program's file is 1DKA100:[DEVELOPMENT.EXAMPLES]DEMO.EXE;15, and the program has the following command line tail:

```
    /LIST  "Here comes Fortran 2000!"  EXPERIMENT:[FORTRAN_2000]TESTFILE.TXT
```

The operating system's parsing rules consider blanks to be delimiters, except within a character context, i.e. a character string delimited by double quotes. If successful, the reference GET_COMMAND_LINE (NARGUMENTS=N, STARTUP_COMMAND=S, COMMAND_LINE=C, COMMAND_ARGUMENTS=A, IARGUMENT_LENGTHS=L, ISTATUS=I) assigns the following values to the arguments:

Argument	Value
N	3
S	1DKA100:[DEVELOPMENT.EXAMPLES]DEMO.EXE;15
C	/LIST "Here comes Fortran 2000!" EXPERIMENT:[FORTRAN_2000]TESTFILE.TXT
A(1)	/LIST
A(2)	Here comes Fortran 2000!
A(3)	EXPERIMENT:[FORTRAN_2000]TESTFILE.TXT
L(1)	5
L(2)	24
L(3)	37
I	1

GET_ENVIRONMENT (NAME, INAME_LENGTH, VALUE, IVALUE_LENGTH, ISTATUS) obtains the value of the named environment variable.

NAME is a scalar of type CHARACTER of any kind that is defined on the processor. It is an INTENT (IN) argument. It contains the name of the environment variable for which the value is required.

INAME_LENGTH is an optional scalar of type INTEGER of any kind that is defined on the processor. It is an INTENT(IN) argument. It contains the length of the significant portion of the NAME argument.

VALUE is a scalar of type CHARACTER of any kind that is defined on the processor. It is an INTENT (OUT) argument. It is assigned the value of the environment variable which is contained in NAME.

IVALUE_LENGTH is an optional scalar of type INTEGER of any kind that is defined on the processor. It is an INTENT(OUT) argument. It contains the length of the significant portion of the VALUE argument.

ISTATUS is an optional scalar of type INTEGER of any kind that is defined on the processor. It is an INTENT (OUT) argument. On return from the subroutine, ISTATUS is assigned the operating system completion status if it is available. Otherwise, the value of ISTATUS is processor dependent.

On some operating systems, it is possible to have more than one entity with the same name simultaneously (by having the name defined at different levels or in different contexts). In such cases, the rules for determining the values of such multiply defined names are defined by the operating system.

Example
Assume that the operating system's status code for "success" is 0, the value of the PATH environment variable is "C:\WINNT40;C:\WINNT40\SYSTEM32;D:\", and the value of E is PATH. If successful, the reference GET_ENVIRONMENT (NAME=E, VALUE=V, ISTATUS=I) assigns the following values to the arguments

Argument	Value
V	C:\WINNT40;C:\WINNT40\SYSTEM32;D:\
I	0

3. Proposed Edits to be Operated on Later

These edits are preliminary and are provided primarily as a basis for discussion. They are with respect to the Fortran 95 Committee Draft, X3J3 / 96-007r1.

Add the following:

13.14.38a GET_COMMAND_LINE ([NARGUMENTS, STARTUP_COMMAND, COMMAND_LINE, COMMAND_ARGUMENTS, IARGUMENT_LENGTHS, ISTATUS])

Description. Obtains the command line and related information from the operating system. The command line includes information which is associated with the program at startup, according to the rules and conventions of the program's operating system and environment. This information includes, but is not limited to, character strings included with the program's startup command. Command line arguments appear with the command which starts the program and are separated from each other by delimiters. The processor shall use the operating system definition of command argument delimiters if it is available. If the operating system definition is not available, the definition of command argument delimiters is processor dependent.

Command argument delimiters are defined in this way in order to allow the processor the maximum amount of flexibility to implement delimiters as may be required by the host operating system. There also is flexibility for the processor to define delimiters in case the operating system does not do so.

Class. Subroutine.

Arguments.
NARGUMENTS (optional)

shall be scalar and of type INTEGER of any kind that is defined on the processor. It is an INTENT (OUT) argument. It is assigned the number of command line arguments which appear with the command which starts the program. If there is no command line or command line arguments, it is assigned the value zero (0).

STARTUP_COMMAND (optional)

shall be scalar and of type CHARACTER of any kind that is defined on the processor. It is an INTENT (OUT) argument. It is assigned the file name of the program which is executing. The value of STARTUP_COMMAND is the same value which would be returned by executing an INQUIRE (FILE=file-name-expr) statement on the program's file name.

COMMAND_LINE (optional)

shall be scalar of type CHARACTER of any kind that is defined on the processor. It is an INTENT (OUT) argument. It is assigned the value of the command line tail, i.e. everything in the command line except for the program name. If there is no command line or command line tail, it is assigned all blanks.

The command line tail need not physically follow the program name. It could come before the program name, both before and after, or be associated with the program name in some other fashion. An example of this last option would be a list of resources and other objects associated with the program at startup in a Graphical User Interface (GUI) operating system.

COMMAND_ARGUMENTS (optional)

shall be a rank one allocatable array of type CHARACTER of any kind which is defined on the processor. It is an INTENT (OUT) argument. Each array element is assigned the respective argument of the command line, in the order in which the argument appears on the command line. If the operating system or context does define a specific order, the order for the command line arguments is processor dependent. Each command line argument is separated from the other arguments by delimiters. If COMMAND_ARGUMENTS is not allocated at the time of a call to GET_COMMAND_LINE, it is allocated with a dimension of NARGUMENTS, and each element has the length of the longest argument. If there is no command line or command line tail and COMMAND_ARGUMENTS is not already allocated, COMMAND_ARGUMENTS has a dimension of one and is assigned all blanks.

On some operating systems, it is possible that the parsed command line arguments are available but the entire unparsed command line is not available. In such situations, it would be standard conforming for a processor to return the command line arguments in COMMAND_ARGUMENTS and, at the same time, return all blanks in COMMAND_LINE. It also would be standard conforming for a processor in such situations to return the concatenation of all of the COMMAND_ARGUMENTS values as the return value of COMMAND_LINE.

IARGUMENT_LENGTHS (optional)

shall be a rank one allocatable array of type INTEGER of any kind that is defined on the processor. It is an INTENT (OUT) argument. Each array element is assigned the length of the corresponding element of COMMAND_ARGUMENTS. Whether the length reported in an element of IARGUMENT_LENGTHS is longer than the last non-blank character of the corresponding element of COMMAND_ARGUMENTS is processor dependent.

The length of some command arguments could be longer than the position of the last non-blank character if some or all of the trailing blanks are significant.

ISTATUS (optional)

shall be scalar of type INTEGER of any kind that is defined on the processor. It is an INTENT (OUT) argument. On return from the subroutine, ISTATUS is assigned the operating system completion status if it is available. Otherwise, ISTATUS is assigned a processor-dependent value.

Example
Assume that the operating system's status code for "success" is 1, the executing program's file is 1DKA100:[DEVELOPMENT.EXAMPLES]DEMO.EXE;15, and the program has the following command line tail:

```
    /LIST  "Here comes Fortran 2000!"  EXPERIMENT:[FORTRAN_2000]TESTFILE.TXT
```

The operating system's parsing rules consider blanks to be delimiters, except within a character context, i.e. a character string delimited by double quotes. If successful, the reference GET_COMMAND_LINE (NARGUMENTS=N, STARTUP_COMMAND=S, COMMAND_LINE=C, COMMAND_ARGUMENTS=A, IARGUMENT_LENGTHS=L, ISTATUS=I) assigns the following values to the arguments:

Argument	Value
N	3
S	1DKA100:[DEVELOPMENT.EXAMPLES]DEMO.EXE;15
C	/LIST "Here comes Fortran 2000!" EXPERIMENT:[FORTRAN_2000]TESTFILE.TXT
A(1)	/LIST
A(2)	Here comes Fortran 2000!
A(3)	EXPERIMENT:[FORTRAN_2000]TESTFILE.TXT
L(1)	5
L(2)	24
L(3)	37
I	1

13.14.38b GET_ENVIRONMENT (NAME [, INAME_LENGTH], VALUE [, IVALUE_LENGTH] [, ISTATUS])

Description. Obtains the value of a named environment variable.

Class. Subroutine.

Arguments.

NAME — shall be a scalar of type CHARACTER of any kind that is defined on the processor. It is an INTENT (IN) argument. It contains the name of the environment variable for which the value is required. If the INAME_LENGTH argument is not present, the significant length is the same length as would be returned by the LEN_TRIM intrinsic.

INAME_LENGTH (optional) — shall be a scalar of type INTEGER of any kind that is defined on the processor. It is an INTENT(IN) argument. It contains the length of the significant portion of the NAME argument.

VALUE — shall be a scalar of type CHARACTER of any kind that is defined on the processor. It is an INTENT (OUT) argument. It is assigned the value of the environment variable which is contained in NAME.

IVALUE_LENGTH (optional) — shall be a scalar of type INTEGER of any kind that is defined on the processor. It is an INTENT(OUT) argument. It contains the length of the significant portion of the VALUE argument.

ISTATUS (optional) — shall be a scalar of type INTEGER of any kind that is defined on the processor. It is an INTENT (OUT) argument. On return from the subroutine, ISTATUS is assigned the operating system completion status if it is available. Otherwise, ISTATUS is assigned a processor-dependent value.

On some operating systems, it is possible to have more than one entity with the same name simultaneously (by having the name defined at different levels or in different contexts). In such cases, the rules for determining the values of such multiply defined names are defined by the operating system.

Example
Assume that the operating system's status code for "success" is 0, the value of the PATH environment variable is "C:\WINNT40;C:\WINNT40\SYSTEM32;D:\", and the value of E is PATH. If successful, the reference GET_ENVIRONMENT (NAME=E, VALUE=V, ISTATUS=I) assigns the following values to the arguments

Argument	Value
V	C:\WINNT40;C:\WINNT40\SYSTEM32;D:\
I	0

[End of Proposed Subroutines]

Add the following terms to the Glossary:

command line : Information which is associated with the program at startup, according to the rules and conventions of the program's operating system and environment. This information includes, but is not limited to, character strings included with the program's startup command.

command line tail : Everything in the command line except for the program name.

operating system : The master control program for a computing system. The operating system manages and allocates hardware and software resources, schedules and controls the execution of other programs, performs input and output to and from all devices, sets the standards for application programs which run on it, defines the permissible user interfaces, and provides a set of functions and subroutines which application programs can call for use of operating system resources.

[End of Proposed Edits]

4. Summary of Vendor-Specific Implementations

Here is a summary of some vendor-specific implementations. This information is adapted from material prepared by David Mattoon as part of X3J3 JOR Item 016.

Digital Fortran (formerly DEC Fortran and VAX Fortran) for OpenVMS requires that the program be invoked as a "foreign command." Subroutine LIB$GET_FOREIGN (COMMAND_LINE, PROMPT, COMMAND_LEN, FLAGS) is called where COMMAND_LINE returns the command line tail, PROMPT displays a prompt for the user to supply a command line tail interactively, COMMAND_LEN optionally returns the length of the command line tail, and FLAGS is an argument for a "utility command collector" not needed for this application.

IBM AIX/6000: GETARG(I, C) is a subroutine where I specifies which command line argument to return (0=program name), C is an argument of type character and will contain, upon return from GETARG, the command line argument. Subroutine GETENV ('ENVNAM', RESULT) stores in character variable RESULT the value of the environmental variable ENVNAM in the profile file of the current directory.

HP has a function IARGC() which returns the number of arguments and a subroutine GETARG (THSARG, ARG) which returns the THSARG-th argument from the built-in ARG array.

Lahey Fortran: Subroutine GETCL returns a string containing the command tail; everything after the command and the blank separator.

Microsoft Fortran: NARGS()is a function which returns the number of arguments. GETARG (THSARG, ARGMNT, STATUS) is a subroutine where THSARG specifies which command line argument is desired (0=program name), ARGMNT is the actual command line argument, and STATUS is a status: if < 0, an error occurred. If > 0, it is the length of the command line argument returned

POSIX: Several vendors have implemented the IEEE POSIX 1003.9 binding to FORTRAN 77. Function IPXFARGC obtains the number of command arguments. Subroutine PXFGETARG (N, ARG, ARGLEN, IERROR) gets the list of command line arguments. PXFGETARG() places the Nth command-line argument in the character string ARG. The significant length of ARG is returned in ARGLEN. The standard also goes on to define some error conditions (argument-too-long and N-out-of-range), as well as the numbering of arguments (0 is the command/verb, 1 is the first argument, etc). Subroutines PXFGETENV and PXFSETENV respectively get and set environment variables. Subroutine PXFCLEARENV clears all environment variables.

5. References

Freedman, Alan. 1995. Computer Glossary: The Complete Illustrated Dictionary, 7th ed. New York, NY: American Management Association. p. 281.

Institute of Electrical and Electronics Engineers (IEEE). 1992. IEEE Standard 1003.9-1992 - IEEE Standard for Information Technology - POSIX FORTRAN 77 Language Interfaces - Part 1: Binding for System Application Interface (API).

International Business Machines Corporation. December 1993. AIX/XL Fortran Compiler/6000 Language Reference Version 3 Release 1. North York, ON: International Business Machines Corporation. pp. 401, 441.

Lahey Computer Systems. 1995. Fortran 90 Language Reference Revision B. Incline Village, NV: Lahey Computer Systems. pp. 131, 263.

X3J3 / 96-004r1, X3J3 Journal of Requirements, Items 016, 040, and 041.

J3 / 97-152, Command Line Arguments & Environmental Variables - Questions & Answers.

[End of J3 / 97-151]