Command Line Arguments & Environment Variables
Technical Specification

by Craig T. Dedo
July 25, 1997

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

- Honoring the straw vote at Meeting 141, the Everything-at-Once model has been dropped in favor of the Iterative model. Consequently, there are now five (5) intrinsic procedures in place of the previous two (2) intrinsic procedures.
- An optional NAME argument has been added to all procedures in order to obtain or specify the names of command arguments in situations where the operating system or environment specifies names for certain command arguments.
- All of the procedures have ISO_ as a prefix.
- This edition does not include examples or normative text edits.

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 proposal 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.

- Ability to adapt to the likely consequences of the ISO/SC22 mandate for internationalization.

- 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 adhere, to the greatest extent possible, to the rules, conventions, and expectations of the host operating system.

- 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 five (5) separate intrinsic procedures with parallel syntax:

- The subroutine ISO_NCOMMAND_ARGUMENTS obtains the number of command arguments.
- The subroutine ISO_GET_COMMAND_ARGUMENT obtains a specified command argument.
- The subroutine ISO_GET_UNPARSED_COMMAND obtains the unparsed command line tail, i.e., everything in the command line except for the program name.
- The subroutine ISO_GET_PROGRAM_NAME obtains the file name of the program which is executing.
- The subroutine ISO_GET_ENVIRONMENT obtains information from the program's environment.

Most of the arguments of these five (5) procedures 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.

An **operating system** is 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.

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.

A program's **command line tail** includes everything in the command line except for the program name.

> 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 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 and assignment of positional order of command line arguments if such a definition is available. If the operating system definition is not available, the definition of command argument delimiters and assignment of positional order for command line arguments is processor dependent.

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

Positional ordering numbers start with 1 and continue consecutively up to the maximum number of command line arguments. If present, the program name is not included in the numbering of command line arguments.

ISO_NCOMMAND_ARGUMENTS (N, ISTATUS) obtains the number of command line arguments.

N is a scalar of type INTEGER. It is an INTENT (OUT) argument. It specifies the number of command line arguments.

ISTATUS (optional) is a scalar of type INTEGER. 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.

ISO_GET_COMMAND_ARGUMENT (N, VALUE, LENGTH, NAME, ISTATUS) obtains the specified command argument.

N is a scalar of type INTEGER. It is an INTENT (IN) argument. It specifies the positional number of the command line argument.

VALUE is an assumed-length scalar of type CHARACTER. It is an INTENT (OUT) argument. It is assigned the value of the command line argument specified by N.

LENGTH (optional) is a scalar of type INTEGER. It is an INTENT (OUT) argument. It is assigned the length of the significant portion of the command line argument specified by N. Whether the length reported LENGTH is longer than the last non-blank character of the specified command line argument 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.

NAME (optional) is an assumed-length scalar of type CHARACTER. It is an INTENT (OUT) argument. It is assigned the name of the command line argument. If the name of the command line argument is not available, it is assigned all blanks.

ISTATUS (optional) is a scalar of type INTEGER. 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.

ISO_GET_UNPARSED_COMMAND (VALUE, LENGTH, NAME, ISTATUS) obtains the command line tail.

VALUE is an assumed-length scalar of type CHARACTER. It is an INTENT (OUT) argument. It is assigned the value of the command line tail. If there is no command line or command line tail, it 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 the VALUE argument of ISO_GET_COMMAND_ARGUMENT and, at the same time, return all blanks in the VALUE argument of ISO_GET_UNPARSED_COMMAND. It also would be standard conforming for a processor in such situations to return the concatenation of all of the VALUE arguments of ISO_GET_COMMAND_ARGUMENT as the return value of the VALUE argument of ISO_GET_UNPARSED_COMMAND.

LENGTH (optional) is a scalar of type INTEGER. It is an INTENT (OUT) argument. It is assigned the length of the significant portion of the unparsed command line tail. Whether the length reported LENGTH is longer than the last non-blank character of the unparsed command line tail is processor dependent.

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

NAME (optional) is an assumed-length scalar of type CHARACTER. It is an INTENT (OUT) argument. It is assigned the name of the command line tail. If the name of the command line tail is not available, it is assigned all blanks.

ISTATUS (optional) is a scalar of type INTEGER. 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.

ISO_GET_PROGRAM_NAME (VALUE, LENGTH, NAME, ISTATUS) obtains the fully-qualified program name.

VALUE is an assumed-length scalar of type CHARACTER. It is an INTENT (OUT) argument. It is assigned the file name of the program which is executing. The value of VALUE is the same value which would be returned by executing an INQUIRE (FILE=file-name-expr) statement on the program's file name. If the program name is not available, the return value of VALUE is all blanks.

LENGTH (optional) is a scalar of type INTEGER. It is an INTENT (OUT) argument. It is assigned the length of the significant portion of the program name. Whether the length reported by LENGTH is longer than the last non-blank character of the program name is processor dependent.

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

NAME (optional) is an assumed-length scalar of type CHARACTER. It is an INTENT (OUT) argument. It is assigned the name of the command line argument. If the name of the program name argument is not available, it is assigned all blanks.

ISTATUS (optional) is a scalar of type INTEGER. 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.

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

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

INAME_LENGTH (optional) is a scalar of type INTEGER. It is an INTENT(IN) argument. It is assigned the length of the significant portion of the NAME argument.

VALUE is a scalar of type CHARACTER. 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. 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. 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.

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.

[The normative edits to Chapter 13 will appear in a future edition of this paper.]

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. History

Meeting 141 - May 1997

Straw Vote: Which model do you prefer?

4	Everything-at-Once (something similar to 97-151)
14	Iterative (something similar to 97-153)
0	No feature at all
1	Undecided

Dedo, Craig. J3/97-151r1, Command Line Arguments & Environment Variables - Everything-at-Once Model. This was a revision of paper J3 / 97-110 and included 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".

Dedo, Craig. J3/97-152, Command Line Arguments & Environment Variables - Questions & Answers.

Maine, Richard. J3/97-153, Proposed Specs and Syntax for System Arguments. This paper proposed the iterative model and was prepared as a counter-proposal to the model proposed in paper J3/97-151r1.

Dedo, Craig. J3/97-163, Command Line Arguments & Environment Variables - Ideas for Straw Votes.

Meeting 140 (Combined J3/WG5) - February 1997

Votes - Add Command Line Arguments to list of Fortran 2000 Requirements

Straw Vote (Subgroup):	Yes - 9	No - 3	
Straw Vote (Full Committee):	Yes - 29	No - 4	
Straw Vote: What Priority	A - 6	B - 15	C - 12
Straw Vote: What Priority		B - 19	C - 13

Included in Additional Minor Technical Enhancements list in paper J3/97-138r1 (WG5-N1259), Content of Fortran 2000

Dedo, Craig. J3/97-110 (WG5-N1242), Command Line Arguments and Environment Variables.

6. 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 / 97-195, Command Line Arguments & Environmental Variables - Questions & Answers.

[End of J3 / 97-201]