Command Line Arguments & Environment Variables
Questions & Answers

by Craig T. Dedo
February 25, 1997

This paper is intended to answer some of the questions which people have raised concerning the particular implementation of Command Line Arguments and Environment Variables which I have proposed in papers X3J3 / 97-110 and J3 / 97-151.

Disclaimer: These questions and answers only represent my own personal opinion. They do not represent in any way any official opinion or position of X3J3, WG5, ANSI, the ISO, or any other organization at all.

Q: Why did you choose this everything-at-once approach instead of simply using the approach (similar to existing implementations by Hewlett-Packard, IBM, and Microsoft) which has an IARGC subroutine to get the number of arguments and a GETARG subroutine to get a particular argument?

A: There are several reasons:

1. I wanted to solve a number of closely related problems at once. I wanted to do the job right the first time, so that we would not have to revisit the issue later.
2. I wanted to put control into the hands of the application programmer, rather than limiting the capabilities which the application programmer has available. An everything-at-once approach provides these capabilities for the application programmer, while the IARGC/GETARG approach does not.
3. I wanted a facility which has the look-and-feel of Fortran 90 and takes advantage of the advanced features of Fortran 90 and Fortran 2000.
4. Human productivity (e.g., function points per labor hour) is one of my major concerns in language design. Because of its inherent adaptability, I believe that the wholistic approach will offer greater productivity and therefore will be more cost effective for the application developer than the IARGC/GETARG approach.
5. Different users have different needs. Even the same user has different needs under different circumstances. Sometimes the user needs an unparsed command line. Sometimes the user needs it parsed. Sometimes the user needs both at the same time. Therefore, an everything-at-once approach which allows for different usages of this feature is preferable to a one-size-fits-all approach.
6. Persons who are comfortable with the IARGC/GETARG approach can still use that method with the wholistic model. They simply can call GET_COMMAND_LINE with only the NARGUMENTS and COMMAND_ARGUMENTS arguments and then iteratively search through the COMMAND_ARGUMENTS array. The reverse is not true.
7. Philosophically, I believe that standards committees should be in the business of providing to users the tools that they need in order to effectively solve their problems. They should not be in the business of limiting the options which users have available, or telling users what is the One True Way to solve their problems. Users, not members of standards committees or outside experts, are usually the best judges of how to effectively solve their problems.

Q: Would this approach be costly for vendors to implement?

A: First, I will qualify my answer by revealing that I do not have any experience implementing any programming language nor do I have access to language development cost information. I invite those with such knowledge to shed light on this issue, to the extent that they can without violating

confidentiality agreements with their employers. That said, I do not believe that this approach would be very costly for vendors to implement. Here are my reasons:

1. Parsing is a well advanced technology. Parsing a command line is a far less complex endeavor than parsing Fortran source code.
2. Most modern operating systems (OS) already provide OS routines to get most or all of the information that this approach requires.
3. Most of the functionality is already required in all C and C++ compilers. A vendor could do a quick-and-dirty implementation by simply calling or duplicating the appropriate C language facilities. Currently, there is a Technical Report on Interoperability with C in the works. I expect that it will be released soon and it is intended to become part of Fortran 2000. I expect that most vendors will implement the Interoperability provisions fairly soon, since many of their customers want this functionality. The Interoperability with C feature will help ease the burden of implementing this approach.

Q: In this proposal, GET_COMMAND_LINE currently has 6 arguments. Isn't this subroutine much too complicated for the average user?

A: Not at all, for these reasons:

1. This feature is intended for Fortran 2000, not FORTRAN 77. Starting with Fortran 90, procedures arguments can be optional. If the programmer uses the keyword form of the subroutine call, the programmer can select the arguments which he or she needs and ignore the information which he or she is not interested in.
2. Psychological research shows that the average person can keep track of up to 7 pieces of information at the same time (Miller 1956; see also McConnell 1993, 108-109). This suggests that it is acceptable to use up to 7 arguments in one subroutine interface.

Q: Why do you want these subroutines to be intrinsic to the language rather than as part of an intrinsic module for operating system services?

A: There are a number of reasons:

1. I want to avoid the problems that C language application programmers have in having to remember which of over a dozen header files a given intrinsic routine is defined in. I do not want Fortran programmers to be faced with the problem of, "Now, if I reference function F I have to remember to USE module M." Application programmers should be spending their time developing their applications, not fighting the intricacies and limitations of the language.
2. The concept of an intrinsic module was developed by John Reid in order to resolve an objection to floating point exception handling. Some persons who wanted execution speed at any price objected to that feature because they thought that the mere presence of exception handling, even if unused in their own program, would slow down their code. The command line argument functionality does not pose such problems.
3. I do not believe that so-called "name space pollution" is as serious an issue as many others mention. We have up to 31 characters available for symbolic names. Even if we do not consider nonsense or objectionable combinations, there are many more combinations available than people could possibly use. In order to avoid name conflicts, we could reserve a prefix, e.g., ISO_, for use by the standards bodies.

Q: Why use operating system definitions for status values and other constructs instead of leaving this completely up to the processor?

A: Some operating systems, such as DEC's OpenVMS, have their own well-defined system of condition codes. DEC's OpenVMS even has a complete Message Utility by which application developers can

develop their own condition codes in a way which does not conflict with the operating system's condition codes. I wanted to define the status values in such a way that they do not conflict with possible operating system condition codes. This same principle is also the reason for deferring to the operating system's conventions and usages for the other issues, such as the order of command line arguments and the definition of command argument delimiters.

Q: Why is it necessary to define the terms "operating system", "command line", and "command line tail"?

A: High quality standards work requires clear, well-defined terms. In general technical usage, these terms are well defined and well understood. Since we are developing features that interact closely with the operating system, it makes sense to use the definitions which already exist.

Q: Why did you choose the argument names that you did? Aren't these names a throwback to the ancient FORTRAN 66 practice of implicit data typing by the first letter of the name?

A: The choice of argument names was not intended to state or imply any support for implicit data typing. To the contrary, the choice of argument names was inspired by Hungarian notation, a naming convention for data objects originated by Charles Simonyi. Although it is used much more frequently in the C programming language rather than in Fortran, Hungarian notation is naiming methodology which is applicable to any modern programming language.

Q: Due to differences in how different operating systems define their user interfaces, isn't this feature likely to work so differently from one operating system to another that it will be fairly useless?

A: Obviously, due to differences between operating systems, a 100% solution is not possible. Refusing to develop this feature means that the decision maker is insisting on a 0% solution. There are 99 other percentage points between 0% and 100%. I firmly believe that an 80% or 90% solution is good enough so that it offers substantial benefit to the application developer.

Q: Many Graphical User Interfaces (GUIs) do not have a traditional command line. Doesn't the lack of a command line make this feature useless in a GUI environment?

A: No. The proposal allows an implementor substantial freedom to adapt this feature to the rules and conventions of the operating system and environment. In a GUI, any information which is associated with the program at startup could be considered to be part of the program's "command line". In addition, many GUI environments, such as the variations of Microsoft Windows, still offer a command line as part of the program's startup definition.

Q: This feature has been presented and rejected many times before. Why raise it again when it has been defeated so many times?

A: There are several reasons to bring it back again.

1. Most importantly, user demand. Many kinds of users want this feature. On comp.lang.fortran, hardly a week goes by that a user asks how to get information from the command line. Each new user who asks for this feature is another request that X3J3 make it available. Members of the US Congress and the various state legislatures have a rule of thumb which states that for every constituent who writes a letter on a particular issue, there are 1000 others who feel the same way but do not take the time and effort to write. Judging by the number of unique requests, there is a large demand for this feature.

2. C and C++ already have had this feature for many years. It is no secret that Fortran has very strong competition from C and C++.

3. At the joint WG5 / X3J3 meeting held February 10-14, 1997, the straw vote on Tuesday, February 11 showed 29 Yes votes and 4 No votes. This shows substantial support at the international level.

Q: Why require the CHARACTER arguments to be of any kind that is defined on the processor? Why not restrict the kind of the CHARACTER arguments to be of default kind?

A: There are a number of reasons for requiring the CHARACTER arguments to be of any kind that is defined on the processor.

1. There is a problem with the definition of the word "default". As used in the standard (and as confirmed by experience) the word "default" is used in two very different ways:
   a. That kind which is most common or usual in the problem to be solved.
   b. That kind which is most natural for the particular processor's architecture.

These two usages need not be the same on a particular processor. That many Fortran processors have a compiler option which allows the user to specify the default kind for integers and/or reals is evidence of these differing usages.

2. Recently, X3J3 and WG5 received a mandate from the ISO that all future language standards shall have adequate support for porting applications from one culture to another with little or no change in the source code. Although some persons disagree, I firmly believe that this mandate includes support for the ISO 10646 character set. ISO 10646 is a multi-octet, unified set of characters which includes all of the world's alphabets, technical symbols for all major professions, and the major CJK (Chinese-Japanese-Korean) ideographs. It appears to me that part of this mandate is support for ISO 10646 characters in all procedures which have CHARACTER arguments.

3. Not all users are Americans or use the Roman alphabet. Already some applications and users use non-Roman characters. This is especially true for users who are in the Far East or who use the conventions of Far Eastern cultures.

4. In the past, X3J3 has had problems with certain features requiring the use of default kinds of data types. Certain I/O operations require default kinds. Among other things, this places limits on the sizes and numbers of records if the default kind of integer is 32 bits or smaller. Another example is the Fortran 90 SYSTEM_CLOCK subroutine. As currently defined, all three of its arguments require the use of default integers. I would very much prefer not to create more problems like these.

Q: Some previous operating systems used keyword forms of command line arguments. Why not define the COMMAND_ARGUMENTS argument as a data structure so that keyword arguments could be used?

A: Like the design of any product, from Big Macs to advanced software systems, those who design programming language features have to go where the market is. I have to design for the future, not the past. The target time frame for Fortran 2000 is the years 2001 - 2012. Which operating systems are likely to have Fortran 2000 compilers in the year 2001 and beyond? These are likely to be the same OSes which already have or soon will have Fortran 90 compilers. According to my interpretation of Michael Metcalf's "Fortran 90/95 Information", current OSes with Fortran 90 compilers are MacOS, MS-DOS (including Windows 95), Windows NT, OpenVMS, and various kinds of Unix. I do not know of any others.

Q: On many kinds of Unix systems, the command language interpreter (CLI) intercepts and parses the command line and expands what it or the operating system consider to be wild card characters, even before the program starts execution (often called globbing). Thus, the executing program never gets to see the unparsed command line. Doesn't this behavior make this feature quite useless?

A: Not at all for these reasons:

1. Portability across many different platforms is one of the two major reasons for doing this. Even if the meaning (semantics) of the subroutines is not exactly the same in each operating environment, the implementations are likely to be close enough so that the subroutine offers value to the application programmer.
2. There are still many operating systems, such as DEC's OpenVMS, Microsoft's MS-DOS, and Microsoft's Windows NT, which are not members of the Unix tribe. These operating systems have their own rules and behaviors.

Q: On Unix systems with OS-level or CLI globbing, how would an implementor properly interpret what the COMMAND_LINE argument should return to the user?

A: If the OS or CLI has mandatory globbing, there are several possible solutions, any of which could be standard conforming:

1. Return all blanks in COMMAND_LINE, since the behavior of the OS or CLI makes the unparsed command line unavailable.
2. Concatenate all of the parsed command line arguments into one long character string and make this concatenated string the returned value of COMMAND_LINE.
3. If possible, use other facilities of the operating system to find out what the user really passed into the OS or CLI.

Reference List

McConnell, Steven C. 1993. *Code Complete: A Practical Handbook of Software Construction*. Redmond, WA: Microsoft Press.

Miller, G. A. 1956. "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.", *The Psychological Review* 63, no. 2 (March):81-97.

X3J3 / 97-151, Command Line Arguments & Environment Variables - Everything-at-Once Proposal

[End of J3 / 97-152]