Design Considerations for Stream I/O

To: J3
From: Craig Dedo
Date: February 9, 2001
Subject: Design Considerations for Stream I/O

Analysis

The two unresolved issues about stream I/O, 127 and 128, suggest that there may be some serious design defects, not just some editorial fixes to be done. This paper discusses these design considerations and attempts to construct a solid foundation for resolving these two issues. It will attempt to provide a rigorous model using a general approach starting with first principles.

Although many of these ideas are well known and some of them are already part of the normative text of Fortran 2000, I believe that it is useful to present them in as well organized and coherent a manner as possible.

It is my position that stream I/O (and all other I/O access methods for that matter) are intimately connected with the host operating system and file system and must take file system properties into full consideration in order to work well.

The original purpose of stream I/O was to allow users of Fortran to access C-style "byte stream" files or, alternatively, files that have no internal record structure. This is explicitly stated in the two WG5 work items, 63 and 63a, proposing stream I/O.

Following are some design principles and assumptions that form the foundation of this analysis.

1. Implementation details should be left to the processor.
2. We should design for all commercially significant operating systems and file systems.
3. Fortran compilers should work well on a wide variety of operating systems and file systems. No one operating system or file system should dominate the design of Fortran.
4. Fortran compilers should honor the standards and conventions of the host operating system and file system(s). If the operating system and file system are silent on an issue, then the Fortran compiler is free to do as it pleases.
5. Operating systems can support more than one file system, often simultaneously on the same system. A good example is Microsoft Windows NT, which can have some disk volumes with the FAT file system and other volumes with the NTFS file system on the same system at the same time.
6. No concept of a file is universal, even though some concepts are very widespread.
7. We do not know what file systems will dominate computing 10 to 20 years from now. There is no guarantee or even a high likelihood that the file systems which are predominant today will continue to dominate computing.
8. In the next 10 to 20 years, we may have commercially significant installable file systems that are designed by parties other than the vendor of the operating system, such as commercial third parties or even by the user through development kits.
9. Stream I/O in Fortran is an access method, not some other kind of file attribute. This is a correct design decision.

We should rigorously distinguish the concepts of access method, record structure (a.k.a. record type), and data format. Although these three concepts are closely related, they really are independent concepts that can be clearly distinguished.

- An access method is the way that the program finds the data in a file. Previous versions of Fortran allowed only two access methods, sequential and direct. J3 is now adding stream access to Fortran 2000.
- A record structure, or record type, is the way that records are organized and marked off from one another. A file system may support more than one record structure. For example, a file system may support variable-length records, fixed-length records and stream records.
- Data format specifies whether the data is read and written using formatted input/output statements or is unformatted.

It may be possible to access a file of a given record structure in more than one way. This possibility is explicitly anticipated in the normative text of the Fortran 2000 draft [161:18-21, 172:8-11].

File systems can vary widely in complexity and internal structure. At one extreme is the Unix-style concept of a file, "A file is nothing more than a stream of bytes.". At the other extreme is the OpenVMS RMS (for Record Management Services) file system, which has a very complex internal file structure and record structure. It may be useful to draw an analogy.

One could consider file systems to be strongly or weakly typed, just like data types in programming languages. In weakly typed languages, data objects are given a data type, but it is relatively easy to look at the data in an object as if it were of another data type without generating an error condition. In contrast, in strongly typed languages, considering a data object to be something other than the data type it was declared to be is an error and generates an error condition.

Similarly, file systems can be classified as weakly typed if the data in the records can be accessed as if it had two or more record structures, e.g., by variable-length records or by stream. There is little or no difference in the record structure. In contrast, in a strongly typed file system, a given record structure is carefully defined and differentiated from other record structures. If a file is created with one kind of record structure and then there is an attempt to access it as if it had a different kind of record structure, an error occurs.

If stream I/O is to work properly in Fortran, it must work equally well on both weakly typed and strongly typed file systems. This means that any characteristic or attribute of a file system that varies from one file system to another needs to be left to the operating system and file system. Hence, any issue which is concerned about such file system attributes is necessarily processor-dependent.

If a file system recognizes more than one internal structure, it may be allowable to read the data in an existing file with a given record structure only by using one particular access method or by using more than one access method. If more than one access method is allowed, the processor may be able to detect the file's record structure or the user may need to specify which record structure is in use through means not specified in the Fortran standard. An example of the latter method of detection would be to use one or more nonstandard I/O keywords which specify the internal file and record structure.

The same problems also exist when creating a file or writing to an existing file. Different access methods may, by default, create files with different internal file or record structures. If the host operating system or file system allows the Fortran compiler to use an access method to write to

more than one kind of file or record structure, then the Fortran compiler must have some way of determining which structure to use.

Here is an example. Consider a hypothetical file system that has at least two different record structures, variable length sequential records and stream records. It is possible for the file system to support all four of the following combinations:

Access Method	Record Structure
Sequential	Variable Length Sequential
Sequential	Stream
Stream	Variable Length Sequential
Stream	Stream

There is no requirement in the Fortran 2000 draft that a Fortran processor using such a file system needs to support all four combinations.

A related issue is how file systems mark the end of a record (EOR). There are many different ways of doing this with operating systems and file systems that are commercially important today. The following table lists the methods used by several file systems today.

File System	**EOR Method**
MacOS	<CR>
Microsoft Windows FAT	<CR><LF>
Microsoft Windows NTFS	<CR><LF>
Unix	<LF>
VMS RMS	Depends on record structure (record type)

Issues

Here is the full text of the two issues related to stream I/O.

Issue 127: I'm not convinced that end-of-file conditions are fully covered for formatted streams. Note that there is no endfile record in a formatted stream (and I doubt we want there to be one). A strict reading of the 2nd sentence of 9.2.3.2 would tell me that it didn't apply because the endfile wasn't a result of reading an endfile record, but that's subtle. I'd suggest explicitly adding something about sequential; didn't do that myself in case someone thinks that this should apply.

What happens when reading a partial record at the end of a file? We say that there may be partial records, but I don't see where we ever say what the effects of such a thing are. If there is a partial record at the end of a file, is it possible to position after it so that it is the previous record? Should, perhaps, reading past the end of a partial record be an error instead of an EOF or EOR condition? Does padding apply to partial records? Some of these questions are probably best answered elsewhere than in 9.2.3.2, but I'll lump them all into one J3 note.

Issue 128: The words in 10.5.3 about linefeeds in A output imply to me that a nonadvancing formatted stream output statement that writes a linefeed as the last character in a stream file will cause there to be an empty and incomplete record at the end of the file. Is this empty incomplete record supposed to be distinguishable from having no record? If so, I wonder how Unix-like systems are supposed to distinguish it. If not, I wonder whether we have it described correctly. Same with / editing, where this was just copied

from. These holes leave me unconvinced that the description of record handling "just works" with formatted stream I/O. This related to unresolved issue 127 about handling of incomplete records.

References

01-007, Fortran 2000 Draft
98-209r2, Specs and Syntax for M.25, Stream I/O
98-211r2, Edits for M.25, Stream I/O
99-110r1, Stream I/O - Suggested Changes (Unresolved Issue 68)
01-102, Changes to List of Unresolved Issues
01-139, Issue 127 - End-of-File in Formatted Stream Files
01-140, Issue 128 - Empty Incomplete Record
Compaq Computer Corporation, *Guide to OpenVMS File Applications*, Chapter 2, "Choosing a File Organization" (Web site: www.openvms.compaq.com:8000/72final/4506/4506_pro)

[End of J3 / 01-119]