Gem #39: Efficient Stream I/O for Array Types

by Pat Rogers—AdaCore

Let's get started...

Ada has the notion of "streams" that are much like those of other languages: sequences of elements comprising values of arbitrary, possibly different, types. Placing a value into a stream is easy using the language-defined "stream attributes". The programmer simply calls the type-specific attribute routine and specifies the stream and the value. For example, to place an Integer value V into a stream S, one could write the following:

```
Integer'Write (S, V);
```

Strictly speaking, S is not a stream but, rather, an access value designating a stream. The Integer'Write routine will convert the value of V into an array of "stream elements" – essentially an array of storage elements – and then put them into the stream designated by S. Actually, placing the bytes into the stream is accomplished by dynamically dispatching to a procedure specific to the stream representation.

Although this discussion is couched in terms of placing values into streams, you should understand that reading values from streams is very similar to writing them and that the same efficiency issue and solution apply.

For composite types, such as array or record types, each component value is individually written to the stream using the approach described above. Consider an array type “A” specifying Integer as the component type. The default version of A’Write will call Integer’Write for each component. Thus, each Integer value is converted to the array of storage elements and written to the stream. This component-driven behavior is necessary because programmers can define their own versions of the stream attributes, and naturally will expect them to be called even when the types in question are used as component types within enclosing array or record types.

But suppose the array type is structurally just a sequence of contiguous bytes, and the component type does not have a user-defined stream attribute defined. In that case, calling the component-specific attribute for each array component is unnecessary and inefficient.

For example, suppose you are working with Military-Standard 1553B for communicating application values between remote devices. Ultimately, Mil-Std-1553B sends and receives 32-word buffers, where each word is an unsigned 16-bit value. Suppose as well that you want to write and read these buffers to and from streams. We can override the stream attributes so that a whole buffer value is written directly to the stream instead of writing it one buffer component at a time.

The buffer type could be declared as follows:

```
   type Buffer is array (1..32) of Interfaces.Unsigned_16;
```

We can then override the stream attributes for type Buffer.

First we declare the routines:

```
   procedure Read_Buffer
      (Stream : not null access Ada.Streams.Root_Stream_Type'Class
       Item   : out Buffer);

   procedure Write_Buffer
      (Stream : not null access Ada.Streams.Root_Stream_Type'Class
       Item   : in Buffer);
```

All such stream attributes have the same formal parameter types, i.e., an access parameter designating the class-wide root stream type defined by the language, and the type to be written to, or read from, that stream.

We then “tie” the routines to the stream attributes for type Buffer, thereby overriding the default versions:

```
   for Buffer'Read  use Read_Buffer;
   for Buffer'Write use Write_Buffer;
```

The language-defined root stream type and array element type are declared in package Ada.Streams:

```
package Ada.Streams is

   type Root_Stream_Type is abstract tagged limited private;

   type Stream_Element is mod 2 ** Standard'Storage_Unit;

   type Stream_Element_Offset is range
     -(2 ** (Standard'Address_Size - 1)) ..
     +(2 ** (Standard'Address_Size - 1)) - 1;

   ...

   type Stream_Element_Array is
      array (Stream_Element_Offset range <>) of aliased Stream_Ele

   procedure Read
     (Stream : in out Root_Stream_Type;
      Item   : out Stream_Element_Array;
      Last   : out Stream_Element_Offset)
   is abstract;

   procedure Write
     (Stream : in out Root_Stream_Type;
      Item   : Stream_Element_Array)
   is abstract;

   ...
end Ada.Streams;
```

The user-defined Read_Buffer and Write_Buffer routines will call these stream-oriented Read and Write procedures (via dynamic dispatching) once for the entire

Buffer array value, instead of calling them once per array component. Both routines are very similar, so we will omit the body of of Read_Buffer for the sake of brevity and show just the implementation of Write_Buffer:

```
procedure Write_Buffer
   (Stream : not null access Ada.Streams.Root_Stream_Type'Class
    Item   : in Buffer)
is
   Item_Size : constant Stream_Element_Offset :=
                  Buffer'Object_Size / Stream_Element'Size;

   type SEA_Pointer is
      access all Stream_Element_Array (1 .. Item_Size);

   function As_SEA_Pointer is
      new Ada.Unchecked_Conversion (System.Address, SEA_Pointer
begin
   Ada.Streams.Write (Stream.all, As_SEA_Pointer (Item'Address)
end Write_Buffer;
```

In the above, we cannot simply convert the value of Item, of array type Buffer, to a value of type Stream_Element_Array, so we work with pointers instead. We define an access type designating a Stream_Element_Array that is the exact size, in terms of Stream_Elements, of the incoming Buffer value. Note the use of the Buffer'Object_Size attribute in that computation. That attribute gives us the size of objects of the type Buffer, a wise approach since in general the size of a type may not equal the size of objects of that type. We can then use unchecked conversion to convert the address of the formal parameter Item to this access type. Dereferencing that converted access value (via .all) gives us a value of type Stream_Element_Array that we can pass to the call to Ada.Streams.Write.

Thus we avoid processing each component of type Buffer, instead writing the entire Buffer value at once. That's a much more efficient approach. As we said earlier, reading values from streams is analogous to writing values to them and only differs in obvious, minor ways. That is true for using the default stream attributes as well as in the implementation of Read_Buffer.

About the Author

Pat Rogers has been a computing professional since 1975, primarily working on microprocessor-based real-time applications in Ada, C, C++ and other languages, including high-fidelity flight simulators and Supervisory Control and Data Acquisition (SCADA) systems controlling hazardous materials. Having first learned Ada in 1980, he was director of the Ada9X Laboratory for the U.S. Air Force's Joint Advanced Strike Technology Program, Principle Investigator in distributed systems and fault tolerance research projects using Ada for the U.S. Air Force and Army, and Associate Director for Research at the NASA Software Engineering Research Center. He has B.S. and M.S. degrees in computer systems design and computer science from the University of Houston and a Ph.D. in computer science from the University of York, England. As a member of the Senior Technical Staff at AdaCore, he specializes in supporting real-time/embedded systems developers, creates and provides training courses, and is project leader and a developer of the GNATbench Eclipse plug-in for Ada. He also has a 3rd Dan black belt in Tae Kwon Do and is founder of the AdaCore club "The Wicked Uncles".

Last Updated: 10/13/2017

Posted on: 6/9/2008

(/get-a-quote)

Products (/products)

GNAT Pro (/gnatpro)

CodePeer (/codepeer)

SPARK Pro (/sparkpro)

QGen (/qgen)

Services (/services)

Support (/support)

Industries (/industries)

Automotive (/industries/automotive)

Avionics (/industries/avionics)

Rail (/industries/rail)

Air Traffic (/industries/atm)

Space (/industries/space)

Defense (/industries/defense)

Resources (/resources)

Books (/books)

Tech Papers (/tech-papers)

Documentation (/documentation)

Webinars (/webinars)

Devlog (/devlog)

Company (/company)

About AdaCore (/company/about)

Careers (/company/careers)

Contact (/company/contact)

Customer Support (/support)

Login to GNAT Tracker (/login)

Expert Support (/support)

Contact Us (/company/contact)

Pricing (/get-a-quote)

News (/news)

Press Releases (/press)

AdaCore in the Press (/in-the-press)

Events (/events)

Inside AdaCore (/newsletter)

Community (/community)

Download (/download)

Getting Started (/get-started)

About Ada (/about-ada)

About SPARK (/about-spark)

Academia (/academia)

Overview (/academia)

Projects (/academia/projects)

Universities (/academia/universities)

GAP Login (/login?mode=gap)

Other AdaCore Sites

The AdaCore Blog (http://blog.adacore.com)

Learn.adacore.com (https://learn.adacore.com)

Make with Ada (http://makewithada.org)

(http://twitter.com/AdaCoreCompany)

 (https://www.linkedin.com/company/39996)
 (https://www.youtube.com/user/AdaCore05)
 (https://github.com/AdaCore)  (/rss)