Subject: Re: are there any s/w eng tools for IDL
Posted by William Clodius on Wed, 26 Feb 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Judith Bachman wrote:
>
> <snip>
> As experienced C/C++ programmers we really miss a
> compiler that can warn that we've messed up a calling sequence or
> done something that's probably dumb as far as data typing goes.

1. Rely extensively on KEYWORDS for procedures and functions with more than one or two arguments. Avoid using optional arguments as it is easy to have off by one errors. By relying on keywords you should avoid messing up the calling sequence.

2. Keywords add significant flexibility that can result in spaghetti code within the called procedure. For those keywords that can influence the control of flow of a procedure, I often find it useful to create an additional procedure that checks consistency of keywords, defines defaults, and sets one or two integer values so that the main control of flow can be described in terms of a single case construct that utilizes the integer values. This procedure should not access keywords that are passed on unchanged to other procedures.

3. The CASE construct is in general clearer, more flexible, and less error prone than the switch construct of C/C++, use it more often than you would C/C++'s switch.

> We are finding that we're spending a lot of time doing "desk
> checking" to catch things that a complier catches. Does anyone
> have a "lint" like program for IDL or are we going to have to
> learn to be VERY careful when we code? Does anyone have
> recommended coding standards that might help. We're using a
> "Hungarian notation" derivative to help keep data typing under
> control - that's been a help.

Try to rely on a functional rather than an imperative style, i.e., define an entity in one small section of code and do not change its meaning afterwards. It is tempting to reuse a name to mean more than one thing in an attempt to save space or processing time. However, any potential change in the type of an object, implicitly requires a check on the types of the results, allocating the new object and deallocating the old, and it is not clear to me that any of these actions are avoided by name reuse. However, these implicit checks can be made more explicit, providing runtime error checking, by specifying the shape of the entity on the left hand side of the object i.e. if A is a two dimensional array

A = function_name(...) generates a new object A

A(*,*) = function_name(...) may generate a new object A, but tells the interpreter that you want the result to be consistent with the current definition of A. The interpreter will check to ensure that the new object is two dimensional, consistent in extent and type with the definition of A before the assignment. Note consistency in type may mean that assigning a FLOAT to a BYTE is reported as an error, but assigning a BYTE to an integer results in an implicit conversion that is not reported as an error. There are variants of this where you assign to subsections of A, etc.

>
> Thanks in advance for any suggestions that folks might have.
> Judith Bachman
> Judith.Bachman@gsfc.nasa.gov

--

William B. Clodius  Phone: (505)-665-9370
Los Alamos Nat. Lab., NIS-2     FAX: (505)-667-3815
PO Box 1663, MS-C323     Group office: (505)-667-5776
Los Alamos, NM 87545          Email: wclodius@lanl.gov