Subject: Re: idl vs. pv-wave
Posted by David Ritscher on Wed, 23 Jul 1997 07:00:00 GMT
View Forum Message <> Reply to Message

William B. Gunter wrote:
> My boss wants a comparison between idl and pv-wave. I know they are
> similar, but what are the main differences between them? Can an idl
> program run under pv-wave with no, minimal or many modifications? What
> are the advantages/disadvantages?

As I am currently using PV-Wave in my cardiology research, I have a few comments to offer. This post is similar to one I sent in response to the same question from Steven Jaffe.

This PV-Wave vs. IDL question is, appropriately, a recurring question here - maybe, if enough good responses come in, the FAQ keepers could add a detailed section on this theme; or perhaps it belongs in a separate document (would anyone care to take over this function?).

I am a long-time IDL and PV-Wave user. The comparison between the two products is currently rather complicated. I try to program so that most of what I do is compatible with both packages (at least, except for the widget part of things). The body of the two languages has remained 90% identical up till the newest release of IDL (5.0). Up till this point the basics of the languages were the same, with only trivial differences, except for:

1. A different widget interface
2. A different set of math extension routines. IDL uses Recipes in C, PV-Wave uses the IMSL routines. After the makers of IMSL and PV-Wave merged, IDL lost access to the IMSL routines and then chose to adopt the Recipes in C routines.
3. Each offers their own point-and-click type of interface for those who don't want to use a command-line programming approach or wish to supplement this approach with some quick-access tools.

IDL 5.0 introduces some elements of object-oriented programming. I have developed a software system for high resolution ECG analysis that is in daily use in our hospital. It contains some 35,000 lines of code. It is not object oriented, and at this point much time and energy is lost repairing things in this code, fixing things that get broken when a new feature is added, etc. From this experience the need for an object-oriented approach has become clear to me. The PV-Wave folks say that they have no current intention of changing the basis of the language so that it would support object-oriented programming. (They speak about object oriented, but this has to do with some pre-packaged tools, not with the basis of the language itself). IDL 5.0 also introduces the ability to deal more directly with pointers, which can be handy when dealing with large data sets,

such as my ECG measurements.

Whether object oriented is important for your application depends on what you'll be doing. If the chief goal is to use the software as a hand calculator with good graphics display capabilities, it is probably irrelevant. If the goal is to design software components to be used and extended by a group of people, then I would consider it crucial.

But speaking of object oriented, the makers of PV-Wave are demonstrating some strides in this direction, namely JWave and JNL. It looks like they have given up on the idea of updating their own language and are instead jumping on the Java bandwagon, and providing tools for Java that provides advantages normally available within PV-Wave. There is a writeup on JWAVE on a Netscape DevEdge page: http://developer.netscape.com/guides/components/
JNL is an extension to Java that provides needed numerical capabilities.

It is a hard time to make a choice between these two branches of the language, since both companies are working hard to try to differentiate themselves. There remain problems within the common core language, and I hope that this competition will lead to solutions to these problems. As someone here recently mentioned, when one write, in either language, "a = 3", one has just defined a TWO BYTE INTEGER variable. This was probably a logical default back when the language was first conceived. This leads to strange things, such as:
print, 850 * 77
    -86
A mechanism for backwards compatibility plus future development needs to be reached; perhaps a system flag, where, for example, !LANGUAGE_LEVEL=1 inserted into old routines would provide backwards compatibility to a specified language version.

Don't forget to look at MATLAB as another possible choice. They are also making strides now, and it looks like they are adding enough new language features that one could now program a real system within MATLAB (before, there were major deficiencies, such as no 2-D arrays, no integer type, etc.). Particularly interesting with MATLAB are the availability of 'compilers' that convert MATLAB code to C or C++ code. They have also recently added some object-oriented features. My initial impression is that They do not seem to be as strongly integrated into the language as do those of IDL. A major advantage of MATLAB is the rich collection of toolboxes, often written by software has been developed under MATLAB; I can send you references, if that's of interest.

I would be interested in hearing from current IDL users as to how sufficient the current object-oriented features are, and what is

missing. Reading through the documentation I see, for example, that polymorphism and overloading aren't there.

Regards,

David Ritscher

--
David Ritscher
Zentralinstitut fuer Biomedizinische Technik Tel: ++49 (731) 502 5313
Albert-Einstein-Allee 47 Fax: ++49 (731) 502 5315
Universitaet Ulm Internet:
D-89069 ULM
david.ritscher@zibmt.uni-ulm.de
Germany