Subject: Re: IDL Virtual Machine information
Posted by JD Smith on Thu, 26 Jun 2003 22:23:22 GMT
View Forum Message <> Reply to Message

On Thu, 26 Jun 2003 14:19:53 -0700, Reimar Bauer wrote:

> JD Smith wrote:
>
>> On Thu, 26 Jun 2003 08:28:47 -0700, Liam Gumley wrote:
>>
>>> "Craig Markwardt" <craigmnet@cow.physics.wisc.edu> wrote in message
>>> news:onptl0g9wi.fsf@cow.physics.wisc.edu...
>>>>
>>>> "Liam Gumley" <Liam.Gumley@ssec.wisc.edu> writes:
>>>> > A particularly interesting feature of IDL 6.0 is the IDL Virtual
>>> Machine,
>>>> > which will allow developers to distribute compiled cross-platform
>>>> > applications that do not require IDL to run.
>>>>
>>>> Liam--
>>>>
>>>> You're right this is an interesting development. This may help get
>>>> more IDL applications distributed, which I'm sure is the goal of RSI.
>>>> But the virtual machine is significantly less interesting to me on
>>>> account of the fact that EXECUTE() is disabled:
>>>>
>>>> http://www.rsinc.com/idl/idlvm_faq.asp#runtime
>>>>
>>>> There are a couple of key places in my code where EXECUTE() is
>>>> integral to the operation of the algorithm, and those would not
>>>> transfer over to the IDL VM.
>>>
>>> I wonder why EXECUTE is not allowed?
>>>
>>>
>> Presumably for the same reason that uncompiled .pro routines can't be
>> run with the VM: it doesn't include the byte-code compiler. I'd
>> suspect that EXECUTE works by calling the very same compiler at
>> run-time. If it did include the compiler, the VM could easily be
>> turned into a full-fledged copy of IDL! The CALL_* routines still work
>> because they are only allowed to call routines which are compiled
>> (either natively in IDL, or in the .sav file itself).
>>
>> That said, there are lots of uses of EXECUTE which are no longer really
>> necessary in IDL 6.0, e.g., building variable-length argument lists of
>> dimensions for various routines (I've noticed Craig using that trick a
>> lot). Since the VM will only run .sav files compiled with IDLv6.0,
>> there's no need to hang onto these old constructions for

>> compatibility's sake. Perhaps people could list their typical uses of
>> EXECUTE and we could consider ways to eliminate them?
>>
>> JD
>
> I did a few days ago a feature request to remove all EXECUTEs from die
> idl standard library.
>
> You can do a grep at rsi/idl/lib. There are some important routines too
> which you can't use if you plan to build a vm.

I took a look in the IDL6.0 beta lib/, and found a couple which might cause trouble: CW_FORM and CW_PDMENU. Interestingly, in all cases I found, this aim is almost always building a list of arguments of unknown length. In the case of CW_PDMENU, you could easily modify it to use a custom-built _EXTRA structure or, even better, rely on the fact that well-designed routines will (should) accept some value for all keywords which is equivalent to never having used the keyword (e.g. "foo" and "foo,HELP=0" are equivalent -- neither should activate HELP).

One interesting approach to this variable argument list length issue is seen in Perl: the argument list is always interpreted as a single list, and can always be passed in as a single list, as individual arguments, or as any combination. Obviously, IDL cannot be retro-fitted to use this type of syntax, but one possibility presents itself: for all IDL programs, if the keyword _ARGUMENT is passed a pointer array, the individual arguments will be mapped to pointer values of the slots in the pointer array (as many as there are). E.g.

```
 pro myroutine, a, b, c
```

could be called as

```
  myroutine,_ARGUMENTS=[ptr_new(12),ptr_new(15),ptr_new([14,15 ])]
```

equivalent to

```
 myroutine,12,15,[14,15]
```

This would *only* be useful in the context of run-time determined arguments, but there it would be invaluable, for instance, neatly solving the macro-replay issue Ben raised last week, i.e. creating a standard mechanism for storing and re-using a variable number of routine arguments. It would also introduce a potential for memory-leak unless care was taken, but despite this, it seems useful. You'd have to use /NO_COPY judiciously for large arrays or data structures (which normally would be passed by reference), ensuring

that you restore the data after the call if necessary. A set of helper routines could easily bundle all arguments as a pointer array, and unbundle them afterwards, freeing the pointers:

```
big=lindgen(50000)
args=bundle_arguments(12,15,big,/NO_COPY)
myroutine,_ARGUMENTS=args
unbundle_arguments,args,void,void,big
```

The other use of EXECUTE I've seen is turning the name of a variable (a string) into that variable's values. There's already a way to do this (though it's technically off-limits):

```
IDL> myvar=1
IDL> print,routine_names("myvar",/fetch)
      1
```

Terrible things will happen to you when you use this forbidden function, such as (more) hair growing out of your ears, but nonetheless, it demonstrates that EXECUTE is not essential for this operation.

Just my 2.e-2USD.

JD
