Subject: Re: 6.3 reactions?
Posted by JD Smith on Tue, 09 May 2006 18:54:35 GMT
View Forum Message <> Reply to Message

On Tue, 09 May 2006 00:23:57 -0400, Richard G. French wrote:

> I hope that someone who really understands all of this IDL Bridge stuff
> and who doesn't use objects for every programming task will provide a
> simple description for civilians of just what this capability can do for
> the everyday user, and what it can't do. I'm having trouble figuring out
> whether this is a cool feature that I'd be crazy not to employ, an easy
> way to try to do parallel processing, or something that would require me
> to reprogram everything I do to obtain the leverage from multiple
> processors. So, all of you Ernest Hemingways out there, how about some
> simple, declarative sentences with an example or two of how this feature
> works and why it is a Good Thing. Many of us will be grateful!

Having not used it, as far as I can tell it's a convenient wrapper around pre-existing functionality. You could already, with IDL versions prior to 6.3, spawn another IDL process, setup a shared memory channel of communication between them, copy variables back and forth, and use an inter-process communication method, with special communication code running on each side, to notify the parent process when the child process is done with processing. Then you could read back some results, and make use of them. It was just a big pain.

The new Bridge just makes that all easy for you... no need to setup shared memory and/or some other access (like a semaphore) to communicate back and forth, no need to poll to find out if a process is done with some calculation. Just use Get/Set to send variables (by copying) back and forth through shared memory, use callbacks to get notified when processing is complete, etc. This would be useful if you have a GUI front-end, which you'd like to keep responsive while another IDL process chugs away on some big calculation. Basically, if you like background processing of widget events, you'll love the bridge, because you can have *anything* be background processed.

I don't know what the overhead for starting this sub-process and setting up shared memory is, so it may only pay for really intensive calculations. It would also be a potential speedup if you have a problem which doesn't make use of large enough arrays (a few hundred thousand elements) to benefit from IDL's native threading on multi-processor machines. Again, the key factor determining how useful this will be is the overhead involved in setting up the child process. Does it really go through the entire rigmarole of contacting the license server, expanding and searching the IDL_PATH, etc., establishing the graphics devices, etc.? If so, it will be too heavy-weight for

speeding up smaller scale operations, but of course will be useful for those 1 min or longer operations which otherwise would have blocked.

One other factor to consider: I presume that if you had a large pile of data in process A, and you wanted to spawn sub-process B to work on that data in the background, you'd end up with *two* copies of that data in memory, one each in the space of processes A and B, unless you specifically remove that data on one side, then the other (e.g. with TEMPORARY). You thought preventing memory leaks was hard with one IDL process...

JD
