Subject: Re: IDL objects (not object graphics) tutorial?
Posted by Benjamin Hornberger on Thu, 24 Nov 2005 04:40:19 GMT
View Forum Message <> Reply to Message

Richard G. French wrote:
> I'd like to learn how to make use of IDL objects.

Not being an expert, I can give a few examples when I found objects useful. If you don't know anything about objects, it might be difficult to grasp in the beginning, but I hope you can get the idea.

Often, you'll read that objects remove the separation between data and methods (meaning analysis algorithms, procedures etc.). For instance, I have a couple data analysis operations which involve reading a raw data file, applying a some operations on the data (which include some external parameters) and writing or plotting out the result.

In classical procedural programming, you would have a routine that reads the data file into a variable. Then you would call one or several functions / procedures on the data, also passing the external parameters, and each of them would return some intermediate result into more variables. Finally, you would have some routines which write the different kinds of output (image, plot, binary file, ...). You have a lot of variables to keep track of. If you analyze several data files at once, they become hard to manage. Also, you have to type a lot since you pass data in and out of routines permanently.

After writing an object for that analysis, the process for the analysis technique 'analysisX' might look similar to this:

```
obj = obj_new('analysisX')
obj -> read_datafile, '/path/to/file'
obj -> set_param, paramA=x
obj -> analyze
obj -> show_image ;; might pop up image
obj -> write_binary, '/path/to/file'
obj_destroy, obj ;; or keep the object if you want to reuse it later
```

At first sight, this might not look that revolutionary, but for complex procedures, this can simplify things a lot since everything is contained in one "object". For instance, if you analyze two files at the same time, you only have to keep track of one more object reference instead of all the variables (which are stored in the object and can be extracted if necessary).

If you write the object accordingly, you can change a parameter and update everything up to the final result with one command, like

```
obj -> set_param, paramA=y, /update
```

Also, it becomes quite easy to write a GUI for this analysis procedure.
In simple cases, the GUI doesn't need any real analysis code. Each
button click or whatever event just has to be translated into calling an
object method.

Another example where objects are useful is compound widgets. While
simple compound widgets can be written with standard widget techniques,
you run into limitations soon. The reason is that you can't extend the
WIDGET_CONTROL procedure for the specifics of you compound widget -- the
only thing you can do is get or set a value. If you write the compound
widget as object, you can do anything you want by calling methods on the
object reference. David Fanning's FSC_FIELD or FSC_DROPLIST are good
examples for that.

For complex widget programs, it even makes sense to write the whole
program as an object. One advantage is that you avoid passing around
your "state" or "info" structure all the time, because every event
handler has direct access to all internal variables in the SELF
structure (if you know how to redirect the event to an object method).
The second advantage is that it is easier to communicate between
separate widget programs -- if they are objects, you just call methods
on each other. If the are not, you usually have to send events, which is
much more cumbersome.

Since I wrote a lot already and it's getting late, I won't dwell on the
syntax of object writing. David's book as well as Ronn Kling's
"Application Development with IDL" have an introduction into objects.
Also, the IDL help files are not that bad. And I'm sure others can
explain much better than me what a class, an instance of an object, a
method and SELF is (I might give a try tomorrow).

Happy Thanksgiving,
Benjamin