Subject: Re: Event generation by object.
Posted by J.D. Smith on Wed, 10 Nov 1999 08:00:00 GMT
View Forum Message <> Reply to Message

ehummel wrote:
>
> Suppose an objectData which contains some data. Another objectOther is
> interested in this objectData.
> I like to have the following situation:
> The data of objectData is modified. This objectData generates an event
> on this modification. The event
> can be received by objectOther if this objectOther has subscribed on
> notifications of objectData.
> Both these objects don't have widgets.
> Question: is there an event mechanism in IDL which makes this possible?

If you're trying to solve a specific problem, and don't want a general solution, simply passing the object reference, saving it, and calling the appropriate method on it is fine.

E.G.

```
pro objectData::ModifyData, data
 self.data=data
 self.objOther->Notify,data
end
```

where of course you have to setup the class variable self.objOther in Init.

However, if you are looking for a generic and flexible solution, I have come up with one. Basically, I define a superclass, ObjMsg, which provides a framework for message driven object communication. These messages can be widget events, pseudo-events, or totally unrelated to widgets. Here's the blurb from the Doc header:

```
; PURPOSE: A superclass to define a common framework for message-driven
;       object communication.  The events will include those which
;       arise from widget activity within the objects, but the
;       formalism is extensible to any generic 'object events'.  Both
;       kinds of events are encapsulated by the term 'messages', and
;       are referred to as "object messages" when handled by the
;       protocol defined in this class.  ObjMsg works by maintaining a
;       pointer to a resizable list of current ObjMsg-derived objects
;       to which to deliver messages.  Each object maintains its own
;       list, and can send any qualified message to the valid objects
;       on its list.  Objects which are no longer valid are
;       automatically removed from the list.  The message recipient
```

; list elements may be of any data type, including structures or
; objects themselves, though they can be (and often are) as
; simple as subscribing object references . Methods to override
; to customize ObjMsg objects are suggested. A given object may
; only have one entry in the list at a time, and thus removal
; will be based only on the object reference, and not other info
; provided within the recipient list elements.

In practice, though not always, one object serves as the "broker" of messages. In this case, that might be your objectData object(s). The mechanism for adding yourself to the recipient list for the actual messages available from other objects is unspecified, and can be as simple or full-featured as you like (e.g. "Subscribe me to all messages of type X but not type Y").

While this may seem somewhat of an abstract class, it's actually quite powerful in specific applications. One incredibly useful example: since all messages, events or otherwise, flow through ObjMsg Methods at some level (since chaining up is encouraged by the framework), you can easily debug complicated programs by seeing exactly which messages are being sent to which other objects at which times.

Anyway, it's just one approach... which probably needs some fine tuning. I haven't been totally happy with the "subscription" process specific ObjMsg subclasses are using.

Hope this gives you some ideas.

JD

--
J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|