Subject: Re: passing parameters from base to base
Posted by JD Smith on Mon, 25 Nov 2002 22:28:32 GMT
View Forum Message <> Reply to Message

On Thu, 21 Nov 2002 16:46:23 -0700, David Fanning wrote:

> Gert (Gert.Van.de.Wouwer@NOS_PAMpandora.be) writes:
>
>> I've been trying to figure this one out for a while. I have 2 bases. If
>> in Mainbase the button Set is pushed, a second base SetParams is
>> called. Stuff happens there and the idea is that if SetParams is
>> killed, a series of numbers go back to Mainbase. Now how can you write
>> this neatly, so that the code for the SetParams can easily be used in
>> other progs? These are my thoughts:
>> I could pass a pointer to SetParams that keeps the desired data, but
>> how does Mainbase knows that
>> SetParams is killed and that it needs to update its fields?
>
> If these are widget programs we are talking about, the simplest way of
> communicating between them is to send events back and forth. Typically,
> when program 1 calls program 2 it "registers" that it would like to
> receive updates if "something important" goes on. In my XColors program,
> for example, any program that wants to know about XColors loading a
> color table will register with either NOTIFYPRO, NOTIFYOBJ, or NOTIFYID
> keywords. Then, depending upon whether it is a procedure, object, or
> widget that you want to receive the notification, XCOLORS will use
> Call_Procedure, Call_Method, or SEND_EVENT (via Widget_Control) to
> notify program 1.
>

This is similar to a system I've developed over the years (and which hopefully will be available in the form of a suite of viewer tools "real soon now"). Instead of sending events, I abstract widget events and other forms of intercommunications among objects as "messages", and provide a framework for sending messages, signing up to receive those messages, etc. I call it all "Object Messaging", and "ObjMsg" is the parent class. Any ObjMsg object can send and receive messages from any other ObjMsg object. Widget programs can talk to non-widget programs, and all communication is treated the same way.

Want to get messages from somebody about new pudding flavors?

```
somebody->MsgSignup,self,/PUDDING_FLAVORS
```

Tired of hearing about pudding flavors?

```
somebody->MsgSignup,self,/NONE
```

etc. The flow of messages is not static; indeed it sometimes changes quite often during run time. At its essence, there is, of course, no fundamental difference between this mechanism and just carefully keeping track of which methods to call on which objects when, but the beauty is, it relieves the program from having to remember all this, and tends to promote smaller, more modular code.

For instance, my "tvDraw" object contains a draw widget, and sends all kinds of message, including simple motion events. At any given time, anywhere from 0 to ~10 different objects are interested in motion events. But tvDraw simply doesn't care about all that:

```
self->MsgSend,/TVDRAW_MOTION
```

is sufficient to get all messages sent to all the relevant parties with no further effort. What this means is that, if later on you write another ObjMsg object which would like to hear about motion events, no new code is required.

The up-the-widget-heirarchy event passing paradigm is simple and useful, but for complex programs in encourages you to put everything in one place. Object Messages essentially breaks free from this paradigm: events and messages can be sent anywhere, at anytime.

Once freed from the shackles of object (and especially widget) intercommunication, you can better resist the urge to include everything but the kitchen sink in single burgeoning piles of code, and write small, focused objects, designed to solve one task well.

Of course, it's all smoke and mirrors until I show some code...

JD
