Subject: passing parameters from base to base
Posted by Gert on Thu, 21 Nov 2002 20:19:18 GMT
View Forum Message <> Reply to Message

Hi,

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?
I looked at the examples for compound widgets (e.g. cw_defroi) and these do
the trick but they do not use xmanager. Is this the only way.

thx for any help,

Gert

Subject: Re: passing parameters from base to base
Posted by Pavel A. Romashkin on Tue, 26 Nov 2002 18:47:37 GMT
View Forum Message <> Reply to Message

This sounds sensible, although I didn't see one piece of information (or
did I miss it?) - how do ObjMsg objects know about each other? There has
to be a place they go to when they want to "sign up". Is the awareness
established at initialization, or in a common block? How do they become
aware of each other?
Cheers,
Pavel

JD Smith wrote:
>
> 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

Subject: Re: passing parameters from base to base
Posted by JD Smith on Thu, 28 Nov 2002 00:14:19 GMT
View Forum Message <> Reply to Message

On Tue, 26 Nov 2002 11:47:37 -0700, Pavel A. Romashkin wrote:

> This sounds sensible, although I didn't see one piece of information (or

> did I miss it?) - how do ObjMsg objects know about each other? There has
> to be a place they go to when they want to "sign up". Is the awareness
> established at initialization, or in a common block? How do they become
> aware of each other?
> Cheers,
> Pavel
>

You put your finger right on the tricky bit (good eye ;). Right now,
my solution is to "introduce" objects which want to communicate to
each other on a higher level. E.g. I might say in my wrapper program:

```
oDraw=obj_new('tvDraw')
oFunk=obj_new('tvFunky',oDraw)
oNother=obj_new('tvNother',oDraw)
```

and oFunk could sign up for the tvDraw messages it needs. Perhaps
oFunk even builds an ObjMsg object of its own and signs up with it
directly. Usually this is perfectly adequate: relations and
introductions can be done at compile time or Init time. This works
well for, e.g., building a viewer shell with all the desired plug-ins,
especially when you need to carry widget bases around to build the
interface inside.

But sometimes you'd like to find new ObjMsg objects to communicate
with at run time (maybe because they didn't exist yet when you were
first made). For this I use one other mechanism: if you have an
ObjMsg object, you can query for other ObjMsg objects on *its* message
recipient list, typically by class name (sort of a "friend of a
friend" thing). E.g., if I knew that my oDraw object probably had
relations with an object of class 'tvColor' that I needed to talk to,
I'd say:

```
oColor=self.oDraw->GetMsgObjs(CLASS='tvColor')
```

I'd probably make it robust in case there's no such object to
communicate with. Objects which have an absolute dependency on each
other to function should probably introduce themselves the "old
fashioned way".

Any suggestions much appreciated. Another option is to publish
"capabilities" or lists of potential messages sent in a common block
of some kind, but that gets complicated with many multiples of the
same type of object on the global stack.

JD

Subject: Re: passing parameters from base to base
Posted by David Fanning on Thu, 28 Nov 2002 17:38:29 GMT
View Forum Message <> Reply to Message

JD Smith (jdsmith@as.arizona.edu) writes:

> You put your finger right on the tricky bit (good eye ;). Right now,
> my solution is to "introduce" objects which want to communicate to
> each other on a higher level. E.g. I might say in my wrapper program:
>
> oDraw=obj_new('tvDraw')
> oFunk=obj_new('tvFunky',oDraw)
> oNother=obj_new('tvNother',oDraw)
>
> and oFunk could sign up for the tvDraw messages it needs. Perhaps
> oFunk even builds an ObjMsg object of its own and signs up with it
> directly. Usually this is perfectly adequate: relations and
> introductions can be done at compile time or Init time. This works
> well for, e.g., building a viewer shell with all the desired plug-ins,
> especially when you need to carry widget bases around to build the
> interface inside.

The object-widget system Dave Burridge and I have developed
is similar. We use the notion of an object hierarchy, almost
identical to a widget hierarchy. Our objects can find each
other to pass messages back and forth by searching through
the hierarchy, recursively if necessary. It is similar to
the "find by name" mechanism in widgets.

When we spawn other top-level bases from our application,
they can be given an "event object", which is simply a list
of objects who would like to know about what they are up do.
Events can be diverted to other objects, or the other objects
can simply be notified that an event has occurred. For example,
our draw widget object has been written in such a way that you
can "plug in" event handler objects that interpret events from
the draw widget in many different ways, allowing us to put the
draw widget in "zoom mode" or "report mode" or "draw mode", etc.

As we work with this system, we find stronger and stronger
parallels with the widget system. In fact, we were wondering
the other day if we shouldn't give our objects unique IDs, just
like widgets, so we can keep a list somewhere (an XManager object,
say) that will allow us to find them even more easily, wherever
they happen to be.

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: passing parameters from base to base
Posted by Pavel A. Romashkin on Mon, 02 Dec 2002 16:49:31 GMT

View Forum Message <> Reply to Message

Aha, I knew it! :-)
Yes, I think this is adequate. And what David is talking about seems
pretty much the same and Ok too, as long as traditional relationships
among programs are preserved, i.e. one is launched from another, or a
known/predictable sequence of launches exists. As long as one can/wants
to *pass* anything to another process, everything seems trivial, and
messaging system (as it appears to me) is a neat way of inter-object communication.
I faced this too but what I wanted is absolutely no assumptions
regarding what will launch when, and I did not want to keep any
notations of what *has* already launched. It should all happen by
itself. As long as one is allowed to have a Common variable, this is
simple to achieve because you have a place to keep your "watchdog"
object, but if you wanted to get by without Commons, it is more
difficult and is not nearly as neat anymore.
I experimented quite a bit with non-destructable (or, rather,
self-recreating if deleted by mistake) objects that maintained links to
themselves and others, sort of being a virus infecting all of my
programs. The object is tiny, so maintaining it was not taxing to the
system. This worked to some degree but I never finished it, had to
actually do some work.
Therefore, I agree that the proposed solutions will have to do :-)
Cheers,
Pavel
P.S. BTW, I think that "relations and introductions can be done at
compile time" is an overstatement. Objects have to exist in order to
link anything to them, because their location in heap memory is not
known until then.

JD Smith wrote:
>
> and oFunk could sign up for the tvDraw messages it needs. Perhaps
> oFunk even builds an ObjMsg object of its own and signs up with it
> directly. Usually this is perfectly adequate: relations and
> introductions can be done at compile time or Init time. This works
> well for, e.g., building a viewer shell with all the desired plug-ins,
> especially when you need to carry widget bases around to build the

> interface inside.
>

... snip ...

>
> Any suggestions much appreciated. Another option is to publish
> "capabilities" or lists of potential messages sent in a common block
> of some kind, but that gets complicated with many multiples of the
> same type of object on the global stack.
>
> JD

Subject: Re: passing parameters from base to base
Posted by Stein Vidar Hagfors H[2] on Mon, 02 Dec 2002 17:05:50 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> writes:

[...]
> Any suggestions much appreciated. Another option is to publish
> "capabilities" or lists of potential messages sent in a common block
> of some kind, but that gets complicated with many multiples of the
> same type of object on the global stack.

Ah, J.D., J.D.!

I was about to send a very useful piece of code to build on for you, but then
I discovered the file header:

;*********** Singleton Abstract Class -- must be inherited
;*******************************
;Written by J.D. Smith
;Added VARIANT code, S.V.H. Haugan
;Added methods ALL_SINGLETONS(), PICK_SINGLETON()

You?!

Isn't what you need (to disguise the ugly common block ;-) exactly a singleton
object which acts more or less like an Xmanager object?

I've also added some code to your singleton object (*years* ago, so if you've
kept modernizing your code it might take a little bit of work to join the two)
to allow "variants": Instead of having to create new classes for each darn
singleton (i.e., write actual code just to have one functionally similar but
conceptually different object), you can create named variants of a singleton,
and each named variant is effectively a new singleton class.

Send me an email if you're interested, you'll have the code in no time (NB:
warranty period is long expired for my additions, I'll try my best if you have
problems, but cannot promise anything ;-)

--
 -- --------------
Stein Vidar Hagfors Haugan
ESA SOHO SOC/European Space Agency Science Operations Coordinator for SOHO

NASA Goddard Space Flight Center,   Tel.: 1-301-286-9028
Mail Code 682.3, Bld. 26, Room G-1,   Cell: 1-240-354-6066
Greenbelt, Maryland 20771, USA.   Fax: 1-301-286-0264
 -- --------------

Subject: Re: passing parameters from base to base
Posted by JD Smith on Mon, 02 Dec 2002 20:19:18 GMT

View Forum Message <> Reply to Message

On Mon, 02 Dec 2002 10:05:50 -0700, Stein Vidar Hagfors Haugan wrote:

```
>  JD Smith <jdsmith@as.arizona.edu> writes:
>
>  [...]
>>  Any suggestions much appreciated.  Another option is to publish
>>  "capabilities" or lists of potential messages sent in a common block of
>>  some kind, but that gets complicated with many multiples of the same
>>  type of object on the global stack.
>
>  Ah, J.D., J.D.!
>
>  I was about to send a very useful piece of code to build on for you, but
>  then I discovered the file header:
>
>  ;*********** Singleton Abstract Class -- must be inherited
>  ;*******************************
>  ;Written by J.D. Smith
>  ;Added VARIANT code, S.V.H. Haugan
>  ;Added methods ALL_SINGLETONS(), PICK_SINGLETON()
>
>  You?!
>
```

As I recall, I wrote that not for any particular need but for
someone's query on the group. Like most such code, I write it quickly
and then forget it ;)

> Isn't what you need (to disguise the ugly common block ;-) exactly a
> singleton object which acts more or less like an Xmanager object?

This would help avoid a common block (though everyone would have to know where that singleton Object Manager is somehow), but it doesn't really help me, since I actually *need* all those objects to be separately valid. The problem really isn't the common block, it's one of developing a capabilities system that is flexible. The basic idea is that widget events work in an "up-the-tree" manner, but for object messages, I need to send messages to anyone at any time, i.e. totally unconstrained. My solution is essentially to hand-craft the channels of communication, which actually works more nicely than you'd expect, except for run-time changes.

In an ideal system, ObjMsg objects would find each other and setup their own intercommunication themselves. You get into lots of chicken-and-egg dependency issues in this case though.

JD

Subject: Re: passing parameters from base to base
Posted by JD Smith on Mon, 02 Dec 2002 20:47:29 GMT
View Forum Message <> Reply to Message

On Mon, 02 Dec 2002 09:49:31 -0700, Pavel A. Romashkin wrote:

> Aha, I knew it! :-)
> Yes, I think this is adequate. And what David is talking about seems
> pretty much the same and Ok too, as long as traditional relationships
> among programs are preserved, i.e. one is launched from another, or a
> known/predictable sequence of launches exists. As long as one can/wants
> to *pass* anything to another process, everything seems trivial, and
> messaging system (as it appears to me) is a neat way of inter-object
> communication. I faced this too but what I wanted is absolutely no
> assumptions regarding what will launch when, and I did not want to keep
> any notations of what *has* already launched. It should all happen by
> itself. As long as one is allowed to have a Common variable, this is
> simple to achieve because you have a place to keep your "watchdog"
> object, but if you wanted to get by without Commons, it is more
> difficult and is not nearly as neat anymore.

The relationships don't need to be the same; in fact, accomodating new code without re-writing the original was the foremost and essential idea of this setup. I can write an entirely different object I hadn't thought of at the time, and link it simply using the published messages (or add another message type if what I need isn't available). It's really a compile-time vs. run-time, *not* a code-time

vs. run-time static communication pattern issue. If the latter were
the case, there'd be no reason to use objects at all.

You may be confusing the system I describe as one for widget events
primarily, with the attendant notion of "launching" that widget vis. a
vis XManager. I have ObjMsg objects which have no widget component
whatsoever. And ones which do have a widget component typically are
introspective: if the widget doesn't exist when it's needed, it's
created.

> P.S. BTW, I think that "relations and introductions can be done at
> compile time" is an overstatement. Objects have to exist in order to
> link anything to them, because their location in heap memory is not
> known until then.

Yes, you have to watch your ordering. You'll notice my oDraw was
instantiated before the others which depend on it. When this is
impossible, a run-time class or name-based lookup can solve this (as
with GetMsgObjs()). Another technique which mitigates this problem in
practice is object composition: if one object has an single-point
dependency on another, the simplest way to organize them is to have
the primary object create its very own instance of the subsidiary (at
which point it is free to setup the relevant communication channels).

If it's a total free-for-all of bizarre interdependencies, you almost
have to establish the channels of communication at run time. Of
course, then you're not at all guaranteed that the conversation won't
devolve into name calling and grunting ;). The ultimate limit of this
is to search for objects to speak with everytime you have something to
say, or something you'd like to hear about. Imagine that what you are
saying or listening for is a motion event, and you'll quickly see this
isn't a good idea ;).

JD

> JD Smith wrote:
>>
>> and oFunk could sign up for the tvDraw messages it needs. Perhaps
>> oFunk even builds an ObjMsg object of its own and signs up with it
>> directly. Usually this is perfectly adequate: relations and
>> introductions can be done at compile time or Init time. This works
>> well for, e.g., building a viewer shell with all the desired plug-ins,
>> especially when you need to carry widget bases around to build the
>> interface inside.
>>
>>
> ... snip ...
>

>
>> Any suggestions much appreciated. Another option is to publish
>> "capabilities" or lists of potential messages sent in a common block of
>> some kind, but that gets complicated with many multiples of the same
>> type of object on the global stack.

Subject: Re: passing parameters from base to base
Posted by Pavel A. Romashkin on Mon, 02 Dec 2002 21:41:57 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
>
> In an ideal system, ObjMsg objects would find each other and setup
> their own intercommunication themselves. You get into lots of
> chicken-and-egg dependency issues in this case though.

This is exactly what I was talking about :-)
Cheers,
Pavel

Subject: Re: passing parameters from base to base
Posted by Stein Vidar Hagfors H[2] on Tue, 03 Dec 2002 17:14:52 GMT
View Forum Message <> Reply to Message

"Pavel A. Romashkin" <pavel_romashkin@hotmail.com> writes:

> JD Smith wrote:
>>
>> In an ideal system, ObjMsg objects would find each other and setup
>> their own intercommunication themselves. You get into lots of
>> chicken-and-egg dependency issues in this case though.
>
> This is exactly what I was talking about :-)

I may not be getting exactly what you're talking about, but this is
exactly where I think a singleton might be useful: You can always
"find" it (it'll be created & initialized if it doesn't exist!) as
long as you know it's name, e.g. (pardon any mis-programming here, I
haven't been keeping up with objects in daily life!)

```
   ;; Locate the message center
   dummy = obj_new("message_center",object=MsgCtr)

   MsgCtr->Register,self,"mainprog",uniqkey  ;; Key allows multiple instances
                                    ;; of mainprog
   ;; The LocateObjMsg could generate a proxy object if ObjMsg
   ;; hasn't registered yet!  The key makes sure we don't pick a
   ;; wrong instance of ObjMsg (that has already been connected
   ;; with another self-class object)

   objmsg = MsgCtr->LocateObjMsg("ObjMsg",uniqkey)

   if objmsg->isproxy() then begin
      objmsg->queue_state,/on           ;; Keep all messages I send
      objmsg->inform_me_when_registered,self,"through_this_method "
   end

   objmsg->send_message,self,"Send me data anytime","through_this_method"
```

Etc, etc.. Now, when "ObjMsg" registers with the message center, it
(MsgCtr) will generate a call to the above object's
"through_this_method" method, with information that the proxy "objmsg"
is now replaced with a true ObjMsg instance, with so-and-so unique
key. It will then dump all the queued messages from the proxy onto
the registering ObjMsg (maybe triggered by ObjMsg calling a
MsgCtr->deliver_pending_messages method, gives you more control over
whether or not the object is fully initialized).

The number of variations on this scheme is endless, and details will differ
according to what kind of messaging system is already implemented. However,
it sketches out a method to deal with chicken-and-egg problems.

One key problem might be how to use the unique keys (e.g. system time
at object creation) to enable creation of multiple independent interlinked
systems (e.g., two display windows with separate sets of linked controls).
Only a half-thought-through solution, but I don't see that you have problems
that cannot be solved by this general approach

--
 -- --------------
Stein Vidar Hagfors Haugan
ESA SOHO SOC/European Space Agency Science Operations Coordinator for SOHO

NASA Goddard Space Flight Center,   Tel.: 1-301-286-9028
Mail Code 682.3, Bld. 26, Room G-1,   Cell: 1-240-354-6066
Greenbelt, Maryland 20771, USA.   Fax: 1-301-286-0264
 -- --------------

Subject: Re: passing parameters from base to base
Posted by JD Smith on Tue, 03 Dec 2002 20:10:27 GMT

On Tue, 03 Dec 2002 10:14:52 -0700, Stein Vidar Hagfors Haugan wrote:

```
> "Pavel A. Romashkin" <pavel_romashkin@hotmail.com> writes:
>
>> JD Smith wrote:
>>>
>>> In an ideal system, ObjMsg objects would find each other and setup
>>> their own intercommunication themselves.  You get into lots of
>>> chicken-and-egg dependency issues in this case though.
>>
>> This is exactly what I was talking about :-)
>
> I may not be getting exactly what you're talking about, but this is
> exactly where I think a singleton might be useful: You can always "find"
> it (it'll be created & initialized if it doesn't exist!) as long as you
> know it's name, e.g. (pardon any mis-programming here, I haven't been
> keeping up with objects in daily life!)
>
>    ;; Locate the message center
>    dummy = obj_new("message_center",object=MsgCtr)
>
>    MsgCtr->Register,self,"mainprog",uniqkey  ;; Key allows multiple
>    instances
>                                              ;; of mainprog
>
>    ;; The LocateObjMsg could generate a proxy object if ObjMsg ;;
>    hasn't registered yet!  The key makes sure we don't pick a ;; wrong
>    instance of ObjMsg (that has already been connected ;; with another
>    self-class object)
>
>    objmsg = MsgCtr->LocateObjMsg("ObjMsg",uniqkey)
>
>    if objmsg->isproxy() then begin
>      objmsg->queue_state,/on          ;; Keep all messages I send
>      objmsg->inform_me_when_registered,self,"through_this_method "
>    end
>
>    objmsg->send_message,self,"Send me data
>    anytime","through_this_method"
>
> Etc, etc.. Now, when "ObjMsg" registers with the message center, it
> (MsgCtr) will generate a call to the above object's
> "through_this_method" method, with information that the proxy "objmsg"
> is now replaced with a true ObjMsg instance, with so-and-so unique key.
> It will then dump all the queued messages from the proxy onto the
> registering ObjMsg (maybe triggered by ObjMsg calling a
> MsgCtr->deliver_pending_messages method, gives you more control over
```

> whether or not the object is fully initialized).
>
> The number of variations on this scheme is endless, and details will
> differ according to what kind of messaging system is already
> implemented. However, it sketches out a method to deal with
> chicken-and-egg problems.
>
> One key problem might be how to use the unique keys (e.g. system time at
> object creation) to enable creation of multiple independent interlinked
> systems (e.g., two display windows with separate sets of linked
> controls). Only a half-thought-through solution, but I don't see that
> you have problems that cannot be solved by this general approach

I see what you mean: using a singleton as sort of the grand matchmaker, for a more organized system for setting up communication.

Really the ObjMsg framework is already designed to do much of this for you (delivering messages to the correct place at the correct time). The major differences between your proposed approach and mine are worth highlighting:

- The ObjMsg system relies on each individual object which is interested in communicating with the world to maintain its own personal list of message recipients, and to find other ObjMsg objects, and sign up for the messages it wants to hear about from them. The latter can be accomplished in a variety of ways: by direct "introduction" (often at a higher level where the objects are created), by "composition" of subsidiary objects, and by "friend of a friend" introductions. One other subtlety: *all* messages are delivered through a single "Message" method.

- The MsgCtr system centralizes *all* messaging objects, and allows you to find any one of them using a class name and unique ID. Presumably, it's a unique ID you already know, or you're back in the same boat. Messaging objects can sign up with it and reveal how they can be contacted. I'm not sure how objects specifically request messages, but nonetheless...

The biggest problem I see with a centralized system is the "dependency hell" so familiar to anyone who does package management. Suppose I have 5 instances of a viewer running. Each little plug-in tool of the viewer is a separate object which needs to contact a 'tvDraw' object to get draw events. In your system, how does it go about doing that? Well, if each tvDraw object had a unique ID, that would work. But how do you distribute all those unique ID's?... and you're back in the same boat again.

Thanks for the suggestions.

JD

Subject: Re: passing parameters from base to base
Posted by Pavel A. Romashkin on Tue, 03 Dec 2002 21:19:07 GMT
View Forum Message <> Reply to Message

JD Smith wrote:

< Snip-snip...

> Suppose I
> have 5 instances of a viewer running. Each little plug-in tool of the
> viewer is a separate object which needs to contact a 'tvDraw' object
> to get draw events. In your system, how does it go about doing that?
> Well, if each tvDraw object had a unique ID, that would work. But how
> do you distribute all those unique ID's?... and you're back in the same
> boat again.

I agree with Stein Vidar, and I felt that this thread will inevidably
end up as a singleton discussion. This is known to have happened before :-)
In fact, most of my own chots at it were exactly using the singleton
systems of various sorts. The cornerstone of singletons is in my opinion
that there is only ONE official way in IDL to have anything at all
visible at the global level - Common blocks. Other than that? Well,
Functions and Procedures are also visible globally, and I used them too
to avoid Common blocks (one example is at
http://www.ainaco.com/idl/idl_library/sobj_new.pro). However, I'd say
that a Common is just as ugly as that :-)
Anyway, having zeroed in on the central message manager (a Singleton
object, or, for that matter, even a pointer - one example at
http://www.ainaco.com/idl/idl_library/single_set.pro) I think the rest,
particularly what JD is asking, is not all that difficult. For what I
tried, I didn't care really about how the messages are passed: they just
had a destination, and the message manager simply passed them along. If
the destination didn't exist, well - too bad. Message just went
unnoticed, to the dismay of the user :-)
More seriously, for my purposes it was sufficient to indicate the
recepient of the message by the mouse click - the receving graphics
window, per say. From then on, the output from the several data
processing widget programs was directed to that target. Then, I wanted
to compare that picture with another - just clicked on another target,
and exchange data with it from then on. And if there was no valid
viewer, a new one was created.
Not extremely flexible but it works.
And, again, one example of self-managing, self aware command line system
using Common blocks is my little Display program, which is the key

building block of all of my recent 2D data plotters. It has no messaging
as such (its much too simple for needing it, as it has no modules like
David's recent system) in it but handles the directing of data input by itself.
Cheers,
Pavel