Subject: Need IDL 8 Help
Posted by David Fanning on Thu, 23 Jan 2014 23:57:41 GMT
View Forum Message <> Reply to Message

Folks,

In an on-going effort to learn more about IDL 8, I've plunged into object graphics, etc. At the moment I am implementing an object messaging system for communication between objects. In the past, I implemented this by creating my own "list" object. I've decided to give this a go with lists and hashes, which seem suitable for this purpose. But, in writing the code, things are getting WAY too complicated, leading me to think I don't understand this very well. I thought lists and hashes were suppose to make code more elegant, not more complex.

So, I'm doing something wrong.

Here is the idea. Each object has a container for "message recipients". When you are interested in getting some communication from an object, you register your interest with the object by asking the object to include you in the message recipient "list" (whatever that is). When you register, you tell it what message you are interested in (a string like CHANGED_COLORS, or IMAGE_ZOOMED) and you tell who you are (an object reference).

When a message is generated, the object looks in its message recipient "list" to see who has registered an interest in a particular "message", and sends the message to that object.

So, here is the problem. I want to store (together) messages and their recipients. I need to make sure I don't store duplicates (same message and recipient in the list). I need to retrieve those recipients that are associated with a particular "message". For example, give me a list of all the recipients who are interested in the message "CHANGED_COLORS" or the message "ANY".

How would you implement these requirement using some of the new features in IDL 8?

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Need IDL 8 Help
Posted by Matthew Argall on Fri, 24 Jan 2014 02:09:31 GMT
View Forum Message <> Reply to Message

I would do something like the following: Create a hash whose keys represent the message that is to be sent/received. The value of the key is a list that will contain each of the objects corresponding to that message. You can then check if an object is already in the list.

```
IDL> theHash = hash()
IDL> obj1 = obj_new('idl_container')
IDL> obj2 = obj_new('idlanroi')
IDL> message = 'CHANGED_COLORS'
IDL>
IDL> theHash[message] = List()
IDL> if (max(theHash[message] eq obj1) eq 0) then theHash['CHANGED_COLORS'] -> add, obj1
IDL> print, theHash[message]
<ObjHeapVar21665(IDL_CONTAINER)>
IDL>
IDL>
IDL> if (max(theHash[message] eq obj1) eq 0) then theHash['CHANGED_COLORS'] -> add, obj1
IDL> print, theHash[message]
<ObjHeapVar21665(IDL_CONTAINER)>
IDL>
IDL> if (max(theHash[message] eq obj2) eq 0) then theHash['CHANGED_COLORS'] -> add, obj2
```

Subject: Re: Need IDL 8 Help
Posted by David Fanning on Fri, 24 Jan 2014 19:22:22 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

>
> I would do something like the following: Create a hash whose keys represent the message that is to be sent/received. The value of the key is a list that will contain each of the objects corresponding to that message. You can then check if an object is already in the list.
>
```
> IDL> theHash = hash()
> IDL> obj1 = obj_new('idl_container')
> IDL> obj2 = obj_new('idlanroi')
> IDL> message = 'CHANGED_COLORS'
> IDL>
> IDL> theHash[message] = List()
> IDL> if (max(theHash[message] eq obj1) eq 0) then theHash['CHANGED_COLORS'] -> add,
obj1
> IDL> print, theHash[message]
> <ObjHeapVar21665(IDL_CONTAINER)>
> IDL>
```

> IDL>
> IDL> if (max(theHash[message] eq obj1) eq 0) then theHash['CHANGED_COLORS'] -> add, obj1
> IDL> print, theHash[message]
> <ObjHeapVar21665(IDL_CONTAINER)>
> IDL>
> IDL> if (max(theHash[message] eq obj2) eq 0) then theHash['CHANGED_COLORS'] -> add, obj2

Thanks, good suggestions. I think I have it working now. Although, I have to admit, I don't really understand this code and the way you use the Max function. But, making the value of the hash a list works pretty well, I think. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Need IDL 8 Help
Posted by Matthew Argall on Fri, 24 Jan 2014 19:56:18 GMT
View Forum Message <> Reply to Message

>> IDL> if (max(theHash[message] eq obj1) eq 0) then theHash['CHANGED_COLORS'] -> add, obj1

> I don't really understand this code and the way you use
> the Max function.

theHash[message] is the list of objects receiving/sending messages. Lists have _overloadEQ methods that compare each entity in the list to the desired quantity. If the maximum in the array of 1's and 0's is 0, then you know that the object is not in the list already.

IDL> message = 'CHANGED_COLORS'
IDL>
IDL> theHash = hash()
IDL> obj1 = obj_new('idl_container')
IDL> obj2 = obj_new('idlanroi')
IDL> obj3 = obj_new('idlgrview')
IDL>
IDL> theHash[message] = list()

```
IDL> theHash[message] -> add, obj1
IDL> theHash[message] -> add, obj2
IDL> print, theHash[message] eq obj3
  0   0
```

Subject: Re: Need IDL 8 Help
Posted by David Fanning on Fri, 24 Jan 2014 20:09:41 GMT
View Forum Message <> Reply to Message

Matthew Argall writes:

> theHash[message] is the list of objects receiving/sending messages. Lists have _overloadEQ methods that compare each entity in the list to the desired quantity. If the maximum in the array of 1's and 0's is 0, then you know that the object is not in the list already.
>
> IDL> message = 'CHANGED_COLORS'
> IDL>
> IDL> theHash = hash()
> IDL> obj1 = obj_new('idl_container')
> IDL> obj2 = obj_new('idlanroi')
> IDL> obj3 = obj_new('idlgrview')
> IDL>
> IDL> theHash[message] = list()
> IDL> theHash[message] -> add, obj1
> IDL> theHash[message] -> add, obj2
> IDL> print, theHash[message] eq obj3
>   0   0

Humm. A overloaded operator is kinda like introducing a paragraph into
the documentation that is written in Serbian, isn't it? ;-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Need IDL 8 Help
Posted by Matthew Argall on Fri, 24 Jan 2014 20:22:27 GMT
View Forum Message <> Reply to Message

> Humm. A overloaded operator is kinda like introducing a paragraph into

> the documentation that is written in Serbian, isn't it? ;-)

I thought so until I tried to write one :-p ... not quite sure how to interpret your sarcasm this time, so here is a [perhaps unnecessary] explanation.

Basically, if you do

[list] EQ [something]

each element of [list] will be checked to see if it is equal to [something]. However, one element in a list is not the same as an element in an array. Building on the previous example,

```
> IDL> message = 'CHANGED_COLORS'
> IDL>
> IDL> theHash = hash()
> IDL> obj1 = obj_new('idl_container')
> IDL> obj2 = obj_new('idlanroi')
> IDL> obj3 = obj_new('idlgrview')
> IDL>
> IDL> theHash[message] = list()
> IDL> theHash[message] -> add, obj1
> IDL> theHash[message] -> add, obj2
> IDL> print, theHash[message] eq obj3
>     0   0
```

```
IDL> theHash[message] -> add, [obj1, obj2]
IDL> print, theHash[message] EQ obj1
   1   0   0
```

Subject: Re: Need IDL 8 Help
Posted by David Fanning on Fri, 24 Jan 2014 20:25:44 GMT

View Forum Message <> Reply to Message

Matthew Argall writes:

```
>> Humm. A overloaded operator is kinda like introducing a paragraph into
>> the documentation that is written in Serbian, isn't it? ;-)
>
> I thought so until I tried to write one :-p ... not quite sure how to interpret your sarcasm this
time, so here is a [perhaps unnecessary] explanation.
```

I appreciate the information. It's helpful. :-)

Cheers,

David
--

David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: Need IDL 8 Help
Posted by kagoldberg on Sat, 25 Jan 2014 08:55:35 GMT
View Forum Message <> Reply to Message

Here's another, related way to do this: broadcast your message to all child objects, and let the child objects decide if that message requires them to take action.

Suppose each message-able graphic object has a 'parent' keyword in the Init method where we pass the parent object's reference. Every such object then knows its parent explicitly. (You could store this information in the fields of the object, or only use it in the Init method.)

In their Init methods, objects call their parent's 'register' method, which simply stores the object reference in a big hash or list. The only argument to 'register' is the child object's reference (i.e. self, when you are within the Init method of the child).

Now, whenever the parent has ANY sort of relevant change (no matter the type) it tells every registered object, with a foreach loop, calling the 'notify' methods of those objects. The parent sends some kind of coded "type of change" or "type of message" information, plus whatever relevant arguments are necessary to interpret the change. (Lots of ways to do this, obviously.)

Then within the 'notify' methods, it's up to each child to determine if the information it is given is important to it, or if it should be ignored. I'd use a case statement in the 'notify' method to select what to do with each possible kind of notification.

You give up a small amount of efficiency by broadcasting messages to objects that may not need to know about them (since you're notifying all registered objects). BUT, you save yourself all of the trouble of having to keep track of which objects need to be the recipients of what kinds of information. That latter system could drive you mad, and I think that's the complexity that drove your question.

Unless you have a huge number of child objects, messaging all of the registered objects will happen so fast.

Another tip with object graphics and this messaging system you're creating is to delay Draw updates until everyone has been messaged, then draw all at once. If every notification prompts a re-draw, you're going to see a big slow-down.

If you code it right, you could make the notification recursive. So when an object is notified, it acts on relevant information, and then passes the notification (or a modified notification) down it ITS registered children. This makes notification distribution hierarchical, rather than flat.
