Subject: widgets and objects
Posted by Matthew Hanson on Thu, 30 Oct 1997 08:00:00 GMT
View Forum Message <> Reply to Message

Hello all,
I am trying to figure out widgets and I don't know if what i am trying to do is even possible - maybe it's just the wrong way to go about it.
I have an object, during the initialization of that object I have a widget come up that has some checkboxes and an OK button for the use to select some choices during initialization. The choices available are dependent on data files read in during initialization and affects what happens next in initialization (data is being read in and the way it is organized is based on the users choices).
Where i am encountering problems is when i call xmanager is still continues with initialization. Do i even want to call xmanager? And how do i pass back the choices that the user chose? I have it set up to call an event when the OK button is pressed. If the choices are valid in the checkboxes it closes the widget and returns the choices to the initialization. There doesn't seem to be an easy way to pass variables around with widget events. Is the only way to do this is by using UVALUE? It looks as if you have to set UVALUE to some variable (structure, vari, or otherwise) and you can then retrieve and set it during the event procedure. But then how can i retrieve it later? When the widget comes up the initialization function finishes. Then the variables that hold the choices go out of scope and are inaccessible to any other procedure.
Do i have this all wrong?
-matt

--
Matthew Hanson

KTAADN, Inc. Phone: (617)527-0054
1320 Centre Street, Suite 201 Fax: (617)527-9321
Newton, MA 02159

matt@ktaadn.com matth@who.net vroomfogle@worldnet.att.net

Subject: Re: widgets and objects
Posted by Reinhold Schaaf on Thu, 13 Aug 1998 07:00:00 GMT
View Forum Message <> Reply to Message

<HTML>
<TT>mirko_vukovic@notes.mrc.sony.com wrote:</TT>
<BLOCKQUOTE TYPE=CITE><TT>Now, the combination of widgets and objects has been mentioned but not</TT>

<TT>described by several folks. Can one define a widget to be an object,

and the</TT>

<TT>events to be methods? Then self is naturally defined, and
at least I do not</TT>

<TT>have to worry about accessing it. The code does not get much
cleaner, but at</TT>

<TT>least one level of pointer access is eliminated. However,
it is not clear to</TT>

<TT>me that the event handlers can be methods. In that sense,
one needs the</TT>

<TT>event handler to call the method, yet another layer of complexity.</TT><TT></TT>

<P><TT>Ideally, when defining the widget, a special type of variable would
be defined</TT>

<TT>that exists in the widget space and in all the event handlers.
It would be</TT>

<TT>like a hidden common block, but would automatically exist in all
the event</TT>

<TT>handlers, and there would be a separate instance of these widget
variables</TT>

<TT>for separately created base widget (the biggest problem with common
blocks).</TT><TT></TT>

<P><TT>Thus, a couple of hints to RSI. Allow the use of methods to
be event handlers</TT>

<TT>and/or allow the creation of these special widget common variables.</TT><TT></TT>

<P><TT>mirko</TT><TT></TT>

<P><TT>-----== Posted via Deja News, The Leader in Internet Discussion
==-----</TT>

<TT><A
HREF="http://www.dejanews.com/rg_mkgrp.xp">http://www.dejanews.com/rg_mkgrp.xp</A>&nbs
p;
Create Your Own Free Member Forum</TT></BLOCKQUOTE>
<TT> </TT>

<TT>I worked quite a bit in this sort of programming and came up with
the following scheme, which was initialized by a remark of Mark Rivers
from the University of Chicago:</TT><TT></TT>

<P><TT>The first part of the scheme is to put a reference of the object
into the widget's UVALUE. The following part of the implementation of the
class CWidget (which is used as an abstract base class for all sorts of
other widget classes) shows how this can be done during initialization.
Whenever the widget is needed, the method CWidget::GetBase returns it,
also for all classes derived from CWidget (unless you override GetBase,
what you should'nt).</TT>

<TT></TT>
<UL><TT>PRO CWidget__Define</TT>

<TT> struct = { CWidget,   &a
mp;nbsp;
$</TT>

<TT>

 wBase:0L,   & amp;nbsp; $</TT>

<TT>

oFrame:OBJ_NEW() $; The object of class CFrame that holds the TLBase</TT>

<TT>
}</TT>

<TT>END</TT><TT></TT>

<P><TT>FUNCTION CWidget::Init, oParent, $
;
; either CWidget or CFrame object</TT>

<TT>

FRAME=bFrame, $; ; set for frame around
widget</TT>

<TT>

XOFFSET=fXOffset, $; in pixels relative to parent</TT>

<TT>

XSIZE=fXSize, $; ; in pixels</TT>

<TT>

YOFFSET=fYOffset, $; in pixels relative to parent</TT>

<TT>

 YSIZE=fYSize   &nbs p;   ; in
pixels</TT><TT></TT>

<P><TT>;some checks deleted</TT><TT></TT>

<P><TT> wParent = oParent->GetBase()</TT>

<TT> self.wBase = WIDGET_BASE(wParent, $</TT>

<TT>

FRAME=bFrame, $</TT>

<TT>

XOFFSET=fXOffset, $</TT>

<TT>

XSIZE=fXSize, $</TT>

<TT>

YOFFSET=fYOffset, $</TT>

<TT>

YSIZE=fYSize)</TT>

<TT> WIDGET_CONTROL, self.wBase, SET_UVALUE=self</TT>

<TT> self.oFrame = oParent-> GetFrame()

; <= it happens here</TT>

<TT> RETURN, 1</TT>

<TT>END</TT><TT></TT>

<P><TT>FUNCTION CWidget::GetBase</TT>

<TT> RETURN, self.wBase</TT>

<TT>END</TT>

<TT> </TT></UL>
<TT></TT>

<P><TT>The second part is event handling: This is all done in a central
event-handling routine, named CFrame_Event, which must be declared as the
event handler in all XMANAGER calls. This is of course a global function,
but its only purpose is to distribute the events to member functions of
CFrame:</TT>

<TT></TT>
<UL><TT>PRO CFrame_Event, sEvent</TT><TT></TT>

<P><TT> stEventName = TAG_NAMES(sEvent, /STRUCTURE_NAME)</TT>

<TT> oFrame = CFrame_GetFrame(sEvent.top)</TT><TT></TT>

<P><TT> bEventHandled = 0B</TT><TT></TT>

<P><TT> CASE stEventName OF</TT><TT></TT>

<P><TT>   ; Events from base widgets</TT>

<TT>   'WIDGET_BASE': $</TT>

<TT>   BEGIN</TT>

<TT>   bEventHandled =
oFrame->OnResize(sEvent.id,
sEvent.x, sEvent.y)</TT>

<TT>   oFrame->OnUpdate</TT>

<TT>   END</TT>

<TT>   'WIDGET_KBRD_FOCUS': $</TT>

<TT>   CASE sEvent.enter OF</TT>

<TT>   0: bEventHandled =
oFrame->OnLooseKbrdFocus(sEvent.id)</TT>

<TT>   1: bEventHandled =
oFrame->OnGainKbrdFocus(sEvent.id)</TT>

<TT>   ELSE:</TT>

<TT>   ENDCASE</TT>

<TT>   'WIDGET_KILL_REQUEST': $</TT>

<TT>   bEventHandled =
oFrame->OnKillRequest(sEvent.id)</TT><TT></TT>

<P><TT>   ;Events from button widgets</TT>

<TT>   'WIDGET_BUTTON': $</TT>

<TT>   CASE sEvent.select OF</TT>

<TT>   ; The frame handles
all events, could be solved differently</TT>

<TT>   0: bEventHandled =
oFrame->OnButtonRelease(sEvent.id)</TT>

<TT>   1: bEventHandled =
oFrame->OnButtonPress(sEvent.id)</TT>

<TT>   ELSE:</TT>

<TT>   ENDCASE</TT><TT></TT>

<P><TT>etc etc</TT></UL>
<TT></TT> <TT></TT>

<P><TT>The class CFrame, which is used as an abstract base class for other
classes, which implement concrete frames, provides only non-functional
event-handlers:</TT>

<TT></TT>
<UL><TT>FUNCTION CFrame::OnResize, wID, iX, iY</TT>

<TT> RETURN, 0B</TT>

<TT>END</TT><TT></TT>

<P><TT>FUNCTION CFrame::OnButtonPress, wID</TT>

<TT> bEventHandled = 0B</TT>

<TT>END</TT><TT></TT>

<P><TT>etc.</TT></UL>
<TT></TT>

<P><TT>The real work is done in the OnResize function of the concrete frame:</TT>

<TT></TT>
<UL><TT>FUNCTION CMainFrame::OnResize, wID, iX, iY</TT><TT></TT>

<P><TT> ;the work is done here:</TT><TT></TT>

<P><TT> lXSizeReq = LONG(iX - self.oDraw->GetXOffset() - self->GetXPad())</TT>

<TT> lYSizeReq = LONG(iY - self.oDraw->GetYOffset() -
self->GetYPad())</TT><TT></TT>

<P><TT> lXSize = self.lXSizeMin > lXSizeReq</TT>

<TT> lYSize = self.lYSizeMin > lYSizeReq</TT><TT></TT>

<P><TT> self.oDraw->SetXSize, lXSize</TT>

<TT> self.oDraw->SetYSize, lYSize</TT><TT></TT>

<P><TT> RETURN, 1B</TT>

<TT>END</TT><TT></TT>

<P><TT>FUNCTION CMainFrame::OnButtonPress, wID</TT><TT></TT>

<P><TT> WIDGET_CONTROL, wID, GET_UVALUE=stButtID ; the ID of the
button pressed</TT>

<TT> CASE stButtID OF</TT>

<TT>   'wButtSetMen': $</TT>

<TT>   BEGIN</TT>

<TT>   IF NOT self->bModeIsSet()
THEN BEGIN</TT>

<TT>   oData =
self->GetData()</TT></UL>
<TT>   etc.</TT><TT></TT>

<P><TT>This scheme has the advantage that all routine work with event handling
is concentrated in the Class CFrame. If one implements a concrete frame
class, one has only to provide the functions CMyFrame::OnResize etc with
the functionality, which is needed for this concrete class. One can forget
all the trouble in CFrame, once it is implemented prooperly!</TT>

<TT></TT>

<TT></TT> <TT></TT>

<P><TT>I hope that my answer gives you some idea how one could procede.
I have developed with this (and some other) concepts the first stage of
a quite complex user interface. But still a lot of work (e.g. with positioning
of widgets) has to be invested into this framework. And so (sad enough,
but that's life), since IDL 5.1 supports ActiveX, I will almost certainly
abbandon widget-object programming in IDL and switch to VC++ + IDL as ActiveX
server.</TT>

<TT></TT> <TT></TT>

<P><TT>Best regards</TT><TT></TT>

<P><TT>Reinhold</TT>

<TT></TT> <TT></TT>

<P><TT>--</TT>

<TT> ** * </TT>

<TT> Reinhold Schaaf</TT>

<TT> Ettighofferstr. 22</TT>

<TT> 53123 Bonn</TT>

<TT> Germany</TT><TT></TT>

<P><TT> Tel.: (49)-228-625713</TT>

<TT> Email: schaaf@astro.uni-bonn.de</TT>

<TT> ** * </TT>

<TT> </TT></HTML>

Subject: Re: widgets and objects
Posted by davidf on Thu, 13 Aug 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Mirko Vukovic (mirko_vukovic@notes.mrc.sony.com) writes:

> this morning, while I was [...] thinking [about] widgets, and what
> is to me the biggest pain, i.e.
> remembering information from one event to the other.

Indeed. But then I'm informed by recent reading that life itself
is pain. Having just had another birthday in the midst of
a week-long tennis tournament, I am prepared to believe it. :-)

> The currently recommended procedure is to use a pointer to a structure which
> contains all the pertinent info. The "aesthetic" problem is that
> you need to recall the pointer from some UVALUE, and then access the contents.

> [Much stuff snipped...]

> Now, the combination of widgets and objects has been mentioned but not
> described by several folks.

Yes, I am working on an example in my "spare" time. :-)

> Can one define a widget to be an object, and the
> events to be methods? Then self is naturally defined, and at least I do not
> have to worry about accessing it. The code does not get much cleaner, but at
> least one level of pointer access is eliminated. However, it is not clear to
> me that the event handlers can be methods. In that sense, one needs the
> event handler to call the method, yet another layer of complexity.

Yes. You can define a widget program to be an object, although
there are some limitations. One is, as you point out, that
event handlers cannot currently be methods. To some extent this
is handled by catching the event in a normal event handler

and calling the event method from there. Not elegant, perhaps,
but it works.

I find the greatest utility of widgets-as-objects to be in
writing compound widgets. The problem with compound widgets is
that it is always difficult to interact with them. For example,
if a compound widget has 15 simple widgets, what does it mean
to "set a value", which is the way you must communicate with
a compound widget.

With a compound widget-as-object, you simply "get the value"
of the compound widget, which is the self reference object,
and you can call any method you like to do anything you like.
Very slick and *very* useful.

I think RSI is aware that widgets as objects are desirable.
Probably if they were redesigning how widgets work that
capability would be built in. As it is now, it is a bit of
a retrofit that could be difficult I imagine. Having just
come off of a major project (object graphics) I can't imagine
there are too many engineers who are ready for another big
push just yet.

In the meantime, I am encouraged by the reports of object
programming reported here and in my own experiments with
it. It would be wonderful if we could start to collect some
of these objects into a library we could share.

Cheers,

David

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

Subject: Re: widgets and objects
Posted by mirko_vukovic on Fri, 14 Aug 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <MPG.103cd43ccb726445989837@news.frii.com>,
davidf@dfanning.com (David Fanning) wrote:
> Mirko Vukovic (mirko_vukovic@notes.mrc.sony.com) writes:
>

>> this morning, while I was [...] thinking [about] widgets, and what
>> is to me the biggest pain, i.e.
>> remembering information from one event to the other.
>
> Indeed. But then I'm informed by recent reading that life itself
> is pain. Having just had another birthday in the midst of
> a week-long tennis tournament, I am prepared to believe it. :-)
>

You now, you are right. Furthermore, as it turns out, you are quoting myself, as I say that to my kids all the time that life is not fair and that it is a pain. And then I tell them to
stop whining, and do something about the problem that is bugging them.

Maybe, just maybe, I should listen to myself (sometimes). :-)

Regarding OOP, I am using quite a bit the concepts introduced by Rumbaugh and Booch. One book, now out of date, but a start is Object Oriented Modeling and Design (Prentice Hall). They have a newer one, much more up to date I am told.

Mirko

Subject: Re: widgets and objects
Posted by mirko_vukovic on Fri, 14 Aug 1998 07:00:00 GMT
View Forum Message <> Reply to Message

In article <35D32FC5.568C67E5@astro.uni-bonn.de>,
Reinhold Schaaf <schaaf@astro.uni-bonn.de> wrote:

!!!!!NOTE

I (Mirko Vukovic) recieved the following e-mail from Reinhold with the request to post it as he had trouble reaching his news-server. I see his message on the group, but in an intelligable font. Thus, I am reposting his email to me.

And now, heeere's REINHOLD!!

I worked quite a bit in this sort of programming and came up with the following scheme, which

was initialized by a remark of Mark Rivers from the University of Chicago:

The first part of the scheme is to put a reference of the object into the widget's UVALUE. The
following part of the implementation of the class CWidget (which is used as an abstract base
class for all sorts of other widget classes) shows how this can be done during initialization.
Whenever the widget is needed, the method CWidget::GetBase returns it, also for all classes
derived from CWidget (unless you override GetBase, what you should'nt).

```
    PRO CWidget__Define
      struct =  { CWidget,          $
                  wBase:0L,         $
                  oFrame:OBJ_NEW()  $ ; The object of class CFrame that
holds the TLBase
                }
    END

    FUNCTION CWidget::Init, oParent, $            ; either CWidget or
CFrame object
                        FRAME=bFrame, $       ; set for frame
around widget
                        XOFFSET=fXOffset, $   ; in pixels relative
to parent
                        XSIZE=fXSize, $       ; in pixels
                        YOFFSET=fYOffset, $   ; in pixels relative
to parent
                        YSIZE=fYSize          ; in pixels

    ;some checks deleted

      wParent = oParent->GetBase()
      self.wBase = WIDGET_BASE(wParent, $
                          FRAME=bFrame, $
                          XOFFSET=fXOffset, $
                          XSIZE=fXSize, $
                          YOFFSET=fYOffset, $
                          YSIZE=fYSize)
      WIDGET_CONTROL, self.wBase, SET_UVALUE=self
      self.oFrame = oParent->GetFrame()                    ; <= it
happens here
      RETURN, 1
    END
```

```
FUNCTION CWidget::GetBase
  RETURN, self.wBase
END
```

The second part is event handling: This is all done in a central event-handling routine, named CFrame_Event, which must be declared as the event handler in all XMANAGER calls. This is of course a global function, but its only purpose is to distribute the events to member functions of CFrame:

```
PRO CFrame_Event, sEvent

  stEventName = TAG_NAMES(sEvent, /STRUCTURE_NAME)
  oFrame = CFrame_GetFrame(sEvent.top)

  bEventHandled = 0B

  CASE stEventName OF

    ; Events from base widgets
    'WIDGET_BASE': $
      BEGIN
        bEventHandled = oFrame->OnResize(sEvent.id, sEvent.x,
sEvent.y)
        oFrame->OnUpdate
      END
    'WIDGET_KBRD_FOCUS': $
      CASE sEvent.enter OF
        0: bEventHandled = oFrame->OnLooseKbrdFocus(sEvent.id)
        1: bEventHandled = oFrame->OnGainKbrdFocus(sEvent.id)
        ELSE:
      ENDCASE
    'WIDGET_KILL_REQUEST': $
      bEventHandled = oFrame->OnKillRequest(sEvent.id)

    ;Events from button widgets
    'WIDGET_BUTTON': $
      CASE sEvent.select OF
        ; The frame handles all events, could be solved differently

        0: bEventHandled = oFrame->OnButtonRelease(sEvent.id)
        1: bEventHandled = oFrame->OnButtonPress(sEvent.id)
        ELSE:
      ENDCASE
```

```
etc etc
```

The class CFrame, which is used as an abstract base class for other classes, which implement
concrete frames, provides only non-functional event-handlers:

```
FUNCTION CFrame::OnResize, wID, iX, iY
  RETURN, 0B
END

FUNCTION CFrame::OnButtonPress, wID
  bEventHandled = 0B
END

etc.
```

The real work is done in the OnResize function of the concrete frame:

```
FUNCTION CMainFrame::OnResize, wID, iX, iY

  ;the work is done here:

  lXSizeReq = LONG(iX - self.oDraw->GetXOffset() - self->GetXPad())

  lYSizeReq = LONG(iY - self.oDraw->GetYOffset() - self->GetYPad())

  lXSize = self.lXSizeMin > lXSizeReq
  lYSize = self.lYSizeMin > lYSizeReq

  self.oDraw->SetXSize, lXSize
  self.oDraw->SetYSize, lYSize

  RETURN, 1B
END

FUNCTION CMainFrame::OnButtonPress, wID

  WIDGET_CONTROL, wID, GET_UVALUE=stButtID ; the ID of the button
pressed
  CASE stButtID OF
    'wButtSetMen': $
      BEGIN
        IF NOT self->bModeIsSet() THEN BEGIN
```

```
        oData = self->GetData()

    etc.
```

This scheme has the advantage that all routine work with event handling is concentrated in the
Class CFrame. If one implements a concrete frame class, one has only to provide the functions
CMyFrame::OnResize etc with the functionality, which is needed for this concrete class. One
can forget all the trouble in CFrame, once it is implemented prooperly!

I hope that my answer gives you some idea how one could procede. I have developed with this
(and some other) concepts the first stage of a quite complex user interface. But still a lot
of work (e.g. with positioning of widgets) has to be invested into this framework. And so (sad
enough, but that's life), since IDL 5.1 supports ActiveX, I will almost certainly abbandon
widget-object programming in IDL and switch to VC++ + IDL as ActiveX server.

Best regards

Reinhold

--


```
 ********************************************************** *
 Reinhold Schaaf
 Ettighofferstr. 22
 53123 Bonn
 Germany

 Tel.:  (49)-228-625713
 Email: schaaf@astro.uni-bonn.de
 ********************************************************** *
```