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>