Subject: Object epiphany: A new way of building widget applications Posted by Martin Schultz on Wed, 04 Apr 2001 21:05:33 GMT

View Forum Message <> Reply to Message

Hi all,

With almost a week delay, I finally get around to release the first version of a new class of IDL objects: the MGS\_GUIObject hierarchy. Don't shy away immediately! It's far easier than it sounds, and once you will have discovered how easy it now becomes to develop widget applications, you will get hooked! Ben Tupper has managed to get something running within a day.

Even though the package is only 68k (mostly documentation;-), I prefer not to attach it to this message but distribute it via anonymous ftp (I may put it on my web site tomorrow). You can get guiobjects\_1-0.zip from ftp://ftp.dkrz.de/pub/Outgoing/martin\_schultz/idl/

Before describing the object hierarchy a little, here is a quick runup over the major features of this new approach:

- (1) You can use the same object program to create blocking, non-blocking, and compound widgets
- (2) In general, you will not have to worry about widget event handling in great depth. Most of this is taken care of for you in the basic MGS\_GUIObject already
- (3) If you compose a widget out of several compound widgets, you simply add them to a built-in container, and they will be managed automatically for you. You only need to overwrite 3-4 methods (with very simple code) to get very powerful widget programs.
- (4) Due to the hierarchical class structure, it is very easy to maintain the code. If you or I want to add a new basic feature to all widget programs, the change only needs to be made once (in MGS\_GUIObject), and all programs will "see" the change immediately.

Most of the programs described below contain an example procedure at the end, so if you ".run mgs\_...\_define" and then call "example, /block" you get to see what they do.

Hierarchy:

MGS\_BaseObject: this is the father (or mother) of all my objects. It contains a pretty sophisticated ErrorMessage interface including traceback information and the choice to display errors as dialogs or on the log window, and including a debug level. This object can also clone itself and do a few other things.

MGS\_Container: this object enhances the IDL\_Container object by allowing object access by name. Compared to the first version of about a year ago, this version now also inherits from MGS\_BaseObject to take advantage of the error handling mechanism.

MGS\_GUIObject: This is the "generic" widget object that lays the foundation for all widget programs to come. It is fully functional in itself, although all you will get is an empty frame with two buttons ("Accept" and "Cancel" or "Apply" and "Close" depending on the Block keyword to the GUI method). The object already handles all major event handling and the general procedure of building and displaying a widget hierarchy. Inherited objects will only need to add the elemental or compound widgets they need in a BuildGUI method; registration with the XManager, positioning of the widget, etc. is all taken care of by MGS\_GUIObject. This object also allows you to set the default font and the label font of your widgets (apparently not on a Mac: Pavel, please test! - and only if you close and rebuild the GUI).

MGS\_Field: This is a sibling of David's FSC\_Field program but extensively modified to make it fit into this new framework and to make it even more modular. As a bonus, I added a character\_mask feature that allows you to specify the characters that shall be allowed for string input. And, special offer for JD;-) you can even specify a regular expression which must then be matched by the input field [needs some tuning, still]. Note, that you can use this field object right away with four statements:

... and if you want to use it as a compound widget, see:

MGS\_InputMask: This widget object simply collects information from a structure and displays the structure tag values (must be scalars) in individual MGS\_Field input fields. Just think of how much work this would be in classical widget programming, then take a look at this code. You'll be amazed!!

MGS\_RangeInput: This object demonstrates how to use two compound

widgets (again MGS\_Field) and link them together to provide a range input pair of fields. I admit they still need some more communication, but you'll get the idea ...

MGS\_Drawcolor: Again, you might recognize the name similarity to David's FSC\_Drawcolor program, and, indeed, that's what this does, too: selecting a color.

MGS\_GUIDemoObject: This object is yet another demonstration of how to combine compound widgets. It builds a widget with three color picker objects and one text field.

Now it's time for me to call the day,

hope you enjoy this new stuff,

Martin

PS: A great big thank you to David again for providing all this great stuff on his web page - and for developing MPI\_Plot which served as a basis for the development of my programs.

Subject: Re: Object epiphany: A new way of building widget applications Posted by davidf on Thu, 05 Apr 2001 18:04:07 GMT View Forum Message <> Reply to Message

Ben Tupper (pemaquidriver@tidewater.net) writes:

> They are pretty quick to cash a check though!

Really!? The last Treasurer's Report shows no receivables for the past 6 months. I'll get back to you, but this may explain Liam's absence. :-(

Subject: Re: Object epiphany: A new way of building widget applications Posted by John-David T. Smith on Thu, 05 Apr 2001 21:57:30 GMT View Forum Message <> Reply to Message

```
Craig Markwardt wrote:
> JD Smith <jdsmith@astro.cornell.edu> writes:
>> Martin Schultz wrote:
>>>
>>> Hi all,
>>>
       With almost a week delay, I finally get around to release the first
>>> version of a new class of IDL objects: the MGS_GUIObject hierarchy.
>>
>> I think it only fair to let people know that I tend to shy away from
>> distributed code with people's initials in the name. I know, it sounds
>> stupid, but I'm not sure I'm the only one. It seems to be a reasonably
>> common practice here (Craig, you listening?), but one which I think
>> might be best to avoid, for the following reasons:
>>
  ... remainder deleted ...
> Hi JD--
> I understand what you are saying, but I think you are a little too
> harsh in criticizing other people for how they name their functions.
> especially when Martin's code is as cool as it sounds.
```

<snip>

Please see my post below. I did not mean to criticize the quality and generosity of, or devotion to the work... not at all. I'm sorry if that's how it came across. I have appreciated Martin's code (and especially his unflagging diligence in documenting it!) as much as anyone else. I always find interesting things reading the comments in his code.

And it appears, since no one has chimed in to say they too prefer more "universal" naming schemes, that I'm the only one whom this bothers. In that case, I'll just keep quiet and deal with it. And at this rate, I can claim the large unoccupied region of the IDL namespace with no initials prepended. My next program will be called "calculate".;)

## Subject: Re: Object epiphany: A new way of building widget applications Posted by davidf on Thu, 05 Apr 2001 22:49:51 GMT

View Forum Message <> Reply to Message

JD Smith (jdsmith@astro.cornell.edu) writes:

> My next program will be called "calculate".;)

Is this JD making a joke!? That thesis must be rocking along! :-)

Cheers.

David

--

David Fanning, Ph.D.

JD Smith wrote:

Fanning Software Consulting

Phone: 970-221-0438 E-Mail: davidf@dfanning.com

Coyote's Guide to IDL Programming: http://www.dfanning.com/

Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Object epiphany: A new way of building widget applications Posted by Paul van Delst on Thu, 05 Apr 2001 22:51:53 GMT

View Forum Message <> Reply to Message

```
> Craig Markwardt wrote:
>>
>> JD Smith <jdsmith@astro.cornell.edu> writes:
>>
>>> Martin Schultz wrote:
>>>> Hi all,
>>>>
>>> With almost a week delay, I finally get around to release the first
>>>> version of a new class of IDL objects: the MGS_GUIObject hierarchy.
>>>
>>> I think it only fair to let people know that I tend to shy away from
```

>>> I think it only fair to let people know that I tend to sny away from

>>> distributed code with people's initials in the name. I know, it sounds

>>> stupid, but I'm not sure I'm the only one. It seems to be a reasonably

>>> common practice here (Craig, you listening?), but one which I think

>>> might be best to avoid, for the following reasons:

>>>

>> ... remainder deleted ...

>>

>> Hi JD--

>>

- >> I understand what you are saying, but I think you are a little too
- >> harsh in criticizing other people for how they name their functions,
- >> especially when Martin's code is as cool as it sounds.

>

> <snip>

>

- > Please see my post below. I did not mean to criticize the quality and
- > generosity of, or devotion to the work... not at all. I'm sorry if
- > that's how it came across. I have appreciated Martin's code (and
- > especially his unflagging diligence in documenting it!) as much as
- > anyone else. I always find interesting things reading the comments in
- > his code.

>

- > And it appears, since no one has chimed in to say they too prefer more
- > "universal" naming schemes, that I'm the only one whom this bothers.

Ah, what the hell: it bugs me too, all this prefixing with initials. If this is equated (in an incorrect, and quite bizarre, manner since the utility or coolness of any particular piece of code has little to do with its name apart from its descriptive purpose) with ones attitude towards - as JD pointed out - code quality, programmer generosity and/or devotion, etc., then I have even less right to state that it bugs me since I don't put myself anywhere even \*near\* the league of the JD's, Schultzs, Fannings, Tuppers, Hadfields, Markwardts, Gumleys etc of the world.

Still bugs me though.

I think, in my case, it's a case of being taught at an early age (in technical writing for scientists writing papers etc.) that use of the first person, e.g. I, we, our, etc, was discouraged. I guess the lesson carried over into other areas. Anyway....

> I'll just keep quiet and deal with it.

Me too.

I read on the train to work today: "Reflective men make uncomfortable prosecutors. By nature and by training, they tend to see the other side and give it equal weight." How's that for some highfalutin fancy-talk?

Now, let's all take a moment to reflect.... :o)

- > And at this rate. I
- > can claim the large unoccupied region of the IDL namespace with no
- > initials prepended.

What the ..!? Me too!

> My next program will be called "calculate".;)

I think you should call it "compute" :o)

Generically Yours,

paulv

--

Paul van Delst A little learning is a dangerous thing;

CIMSS @ NOAA/NCEP Drink deep, or taste not the Pierian spring;

Ph: (301)763-8000 x7274 There shallow draughts intoxicate the brain,

Fax:(301)763-8545 And drinking largely sobers us again.

paul.vandelst@noaa.gov Alexander Pope.

Subject: Re: Object epiphany: A new way of building widget applications Posted by m.hadfield on Thu, 05 Apr 2001 23:38:10 GMT

View Forum Message <> Reply to Message

From: "Paul van Delst" <paul.vandelst@noaa.gov>

- > I think, in my case, it's a case of being taught at an early age (in technical writing for
- > scientists writing papers etc.) that use of the first person, e.g. I, we, our, etc, was
- > discouraged. I guess the lesson carried over into other areas. Anyway....

That does it! From now on all my routine names will prefaced with the letters

## **TOFKAMGH**

O = oceanographer. (I wouldn't be brave enough to call myself one when other oceanographers are present, but I reckon I can get away with it on this group.)

---

Mark Hadfield m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield National Institute for Water and Atmospheric Research

--

Posted from clam.niwa.cri.nz [202.36.29.1]

Subject: A third-party example
Posted by btt on Mon, 09 Apr 2001 13:07:52 GMT
View Forum Message <> Reply to Message

Hello.

Liam suggested that I post the GUI object I made with Martin's GUIobject. I have pasted it below.

Make an instance of the object:
 obj = Obj\_New('HBB\_TwoList') (set Debug = 1 for error traceback)

2. Realize the GUI obj->GUI

- a. Select any item from the left hand (primary) list.
- b. Click the 'copy' button to move it to the right hand (secondary) list.
- c. Items in the secondary list may likewise be selected and removed...
- d. Close the dialog.
- Add the option to move items in the secondary list up and down.
   obj->SetProperty, UPDOWN = 1
   obj->GUI
- a. Note that the earlier selections you made are preserved.
- b. Select an item in the secondary list (add items to the list if there are less than two items to choose from).
- c. Click the MoveUp or MoveDown buttons. Note, the list wraps.

Note that this object overrides the following methods found in MGS\_GUIObject: INIT, CLEANUP, GETPROPERTY and SETPROPERTY.

Martin has made a number of suggestions including:

- Change the basic contruction of the lists so that each can be regarded as indivdual compound widgets... these would be stored in the 'COMPOUND' container
- Permit moving items out of the primary list into the secondary list (rather than simply copying.)
- Override the GUI method so that the MoveUp/MoveDown buttons can be mapped/unmapped rather than simple desensitized. (Although he now has the NOTIFY\_REALIZE keyword in his future.)

- A number of items pertaining to button/list sizing and button names. (Some of which I have tried to include this morning.)

All other suggestions/comments gladly accepted.

```
:-----START HERE-----
;+
: NAME:
; HBB_TwoList
PURPOSE:
The purpose of this widget/object to provide the user an interactive
means of
; of adjusting the contents of a list. Items from a 'primary' list may
be copied to a
'secondary' list. Items in the secondary list may be removed or 'moved';
up' or
;'moved down' relative to each other.
 REQUIREMENTS:
 INHERITS MGS_GUIObject
 CATEGORY:
 Widgets.
 CALLING SEQUENCE:
 objref = OBJ NEW('HBB TwoList')
 ARGUMENTS:
 None required
INITIALIZATION KEYWORDS: (see MGS GUIobject)
 P_LIST (Get/Set) An vector of values for the primary list.
 The datatype is maintained as provided, but the List Widget
 requires that the values be string. So most numeric types
 are acceptable when converted to STRING. The items in the
 the secondary list are selected from this list. If not provided,
 a dummy list (list of dummies?) is provided automatically.
 S LIST (Get/Set) A vector of items for the secondary list. Default = ["]
P_LABEL (Get/Set) A label for the primary list. Default = "
S LABEL (Get/Set) A label for the secondary list. Default = "
; MAX_LENGTH (Get/Set) The maximum number of rows shown in the list,
Default = 30
; MIN LENGTH (Get/Set) The minimum number of rows shown in the list,
```

```
Default = 10
: UPDOWN (Get/Set) Set this keyword to permit up/down shuffling of items
in
; the secondary list using the MoveUp/MoveDown buttons.
MULTIPLE (Get/Set) Set this keyword to permit multiple-item selection
 in either list.
 NO_DUPLICATES (Get/Set) Set this keyword to prevent duplicate
 items from the primary list from appearing in the secondary list.
 EXAMPLE:
o = Obj New('HBB TWOLIST',/UPDOWN)
o->GUI
 ;fiddle with the lists... over items left/right/up/down.
 ;close the dialog
 o->GUI
 ;note your earlier chioces are preserved
 Obj Destroy, o
 ;cleanup
 MODIFICATION HISTORY:
 written 3 APR 2001 Ben Tupper
 Bigelow Laboratory for Ocean Science
 btupper@bigelow.org
 Based upon PRC_DualList widget.
 4 APR 2001 Fixed bug when moving the top item DOWN. BT
 Changed the keyword SHUFFLE to UPDOWN for clarity.
 9 APR 2001 Changed 'Add' to 'Copy' and placed UpDown buttons under the
 secondary list. Forced buttons to same size. BT
; Copyltem
FUNCTION HBB_TwoList::CopyItem, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
p select = widget info(self.listid[0],/List select)
```

If p select[0] NE -1 then begin

```
;if duplicates are not permitted, then
 ;check for duplication otherwise skip this step
If Self.No_Duplicates EQ 1 Then Begin
 A = where(*Self.list[1] EQ (*self.list[0])[p_select], count)
 If Count NE 0 then Return, 0
 EndIf
if n_elements(*self.List[1]) GT 1 Then Begin
 *self.List[1] = [*self.list[1], (*self.List[0])[p_select]]
 EndIf Else begin
 if (*self.list[1])[0] EQ " Then $
 *self.List[1] = (*self.List[0])[p_select] Else $
 *self.List[1] = [*self.list[1], (*self.List[0])[p_select]]
 EndELSE ;the secondary list is empty
Widget_control, Self.ListID[1], Set_Value = String(*Self.List[1])
EndIF; the p_select[0] ne -1
Return,0
END ;Copyltem event
: Removeltem
FUNCTION HBB TwoList::RemoveItem, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
; if the secondary list is empty, then return immediately
If n elements(*(Self.List)[1]) EQ 0 then Return,0
s select = widget info(self.listID[1], /List select)
If s_select[0] NE -1 Then Begin
OldList = *(Self.List)[1]
If n_elements(OldList) GT 1 Then Begin
 index = replicate(1L, n_elements(OldList))
 Index[s\_select] = 0
 A = where(index EQ 1, count)
 if Count gt 0 then *(Self.List)[1] = oldList[a] Else $
```

```
*(self.List)[1] = "
 EndIf Else begin
 *(self.List)[1] = "
 EndElse; OldList only had one item to remove
Widget_control, Self.ListID[1], Set_Value = STRING(*(Self.List)[1])
EndIf;s_select[0] ne -1
Return, 0
END ;RemoveItem event
; MoveUp
FUNCTION HBB_TwoList::MoveUp, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return.0
EndIF
s = widget_info(Self.ListID[1], /List_select)
If s[0] EQ -1 Then Return,0
;return if the list is 'empty'
if (*(Self.List)[1])[s[0]] EQ " Then Return,0
nsel = n elements(s)
nList = n elements(*(Self.List)[1])
;can't allow a higher index than elements in s list
if s[nsel-1] GT (nlist-1) Then Return,0
if s[0] EQ 0 Then Begin ;move the top to the bottom
*(self.list)[1] = Shift(*(self.list)[1], -1)
EndIf Else begin
Index = Lindgen(Nlist)
Index[(s[0]-1L) : (s[nSel-1])] = SHIFT(Index[(s[0]-1L) : (s[nSel-1])], -1)
(*(Self.List)[1])= (*(Self.List)[1])[Index]
EndElse
Widget_control, Self.ListID[1], Set_Value = STRING( *(self.List)[1] )
Return, 0
END ; Move Up event
:-----
; MoveDown
```

```
FUNCTION HBB TwoList::MoveDown, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return.0
EndIF
s = widget_info(Self.ListID[1], /List_select)
If s[0] EQ -1 Then Return,0
;return if the list is 'empty'
if (*(Self.List)[1])[s[0]] EQ " Then Return,0
nsel = n_elements(s)
nList = n_elements(*(Self.List)[1])
;can't allow a higher index than elements in s list
if s[nsel-1] GT (nlist-1) Then Return,0
If s[nsel-1] EQ nList-1 then Begin
*(self.list)[1] = Shift(*(self.list)[1], 1) ;move the bottom to the top
EndIf Else begin
Index = Lindgen(Nlist)
If nsel Eq 1 then $
 Index[s[0]:s[0]+1L] = Shift(Index[s[0]:s[0]+1L],1) Else $
 Index[(s[0]-1L) : (s[nSel-1] + 1L)] = SHIFT(Index[(s[0]-1L) :
(s[nSel-1] + 1L)],1)
(*(Self.List)[1])= (*(Self.List)[1])[Index]
EndElse
Widget_control, Self.ListID[1], Set_Value = STRING( *(self.List)[1] )
Return, 0
END ;MoveDown event
; ListSelect - doesn't do anything (yet) except receive events
FUNCTION HBB_TwoList::ListSelect, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
```

```
END ;ListSelect
:----
: BuildGUI
PRO HBB TwoList::BuildGUI
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return
EndIF
colbase = widget base(self.layoutid, /base align center, column = 3)
width = (MAX(STRLEN(*self.list[0]))+5)>20
Height = self.ListLength[0] > (n_elements(*(Self.List)[0])+3) < Self.ListLength[1]
;column base for Primary List
PID = widget base(colbase, /base align center, column = 1, frame = 1)
pLabel = widget_Label(PID, Value = self.LabelNames[0], /align_center)
pList = Widget_List(PID, Value = STRING(*(Self.List)[0]), $
ySize = height,xsize = width, $
Multiple = Self.Multiple, $
uValue = {Object:Self, Method: 'ListSelect' })
list geo = widget info(pList, /Geometry)
ButtonBase = Widget Base(colbase, /base align center, column = 1, space
= 10)
LabelID = Widget_Label(ButtonBase, value = 'Modify '+self.labelnames[1]+
'Field',$
/align_center)
Move = '>>> Move >>>'
Copy = '>>> Copy >>>'
Remove = '<< Remove <<'
MoveUp = ' Move Up '
MoveDown = ' Move Down '
TransferID = Widget_Base(ButtonBase, Column = 1, /base_align_center)
AddID = Widget_Button(TransferID, Value = copy,$
 uValue = {Object:self, Method:'CopyItem'})
But geo = widget info(AddID, /Geometry)
```

Return, 1

```
RemoveID= Widget_Button(TransferID, Value = remove,$
 xsize = but geo.xsize, $
 uValue = {Object:self, Method:'RemoveItem'})
SID = widget Base(ColBase, /Base align center, Column = 1, Frame = 1)
sLabel = widget_Label(SID, Value = self.LabelNames[1], /align_center)
:create the list with or without a value
if n_elements(*(self.list)[1]) NE 0 then $
sList = Widget List(SID, Value = STRING(*(Self.List)[1]), $
 ysize = list_geo.scr_ysize, xsize = list_geo.scr_xsize, $
 Multiple = Self.Multiple, $
 uValue = {Object:Self, Method: 'ListSelect'}) Else $
sList = Widget_List(SID, $
 vsize = list geo.scr vsize, xsize = list geo.scr xsize, $
 Multiple = Self.Multiple, $
 uValue = {Object:Self, Method: 'ListSelect' })
Self.ListID = [pList, sList]
UpID = Widget_Button(SID, Value = moveup,$
 xsize = but_geo.xsize, $
 uValue = {Object:self, Method:'MoveUp' }, $
 sensitive = self.UPDOWN)
DownID = Widget Button(SID, value = movedown,$
 xsize = but geo.xsize, $
 uValue = {Object:self, Method:'MoveDown' }, $
 sensitive = self.UPDOWN)
END; BuildGUI
; GetValue
FUNCTION HBB_TwoList::GetValue, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
Value = {S LIST: *Self.List[1]}
```

```
GetState
FUNCTION HBB_TwoList::GetState, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch. /Cancel
Return,0
EndIF
If Widget_Info(self.ListID[1], /Valid_ID) Then $
S = Widget Info(self.ListID[1], /List Select) Else $
S = -1
State = {S_LIST:*Self.List[1], S_Select:S}
Return. State
END ;GetState
GetProperty
PRO HBB_TwoList::GetProperty, P_list=P_List, S_list = S_List, $
P Label = P Label, S Label = S Label, $
UPDOWN = UPDOWN, Multiple = Multiple, $
Min_Length = Min_length, Max_length = Max_Length, $
Ref Extra = Extra
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return
EndIF
If arg present(p list) then P List = *(Self.List)[0]
If Arg_present(S_LIST) Then s_List = *(Self.List)[1]
P_Label = Self.LabelNames[0]
S_label = Self.LabelsNames[1]
UPDOWN = Self.UPDOWN
Multiple = Self.Multiple
```

```
No Duplicates = Self.No Duplicates
Min_Length = Self.ListLength[0]
Max_length = Self.ListLength[1]
Self->MGS_GUIObject::GetProperty, _Extra = Extra
END ;SetProperty
SetProperty
PRO HBB_TwoList::SetProperty, P_list=P_List, S_list = S_List, $
P_Label = P_Label, S_Label = S_Label, $
UPDOWN = UPDOWN, Multiple = Multiple, $
Min Length = Min length, Max length = Max Length, $
No_Duplicates = No_Duplicates, $
Ref Extra = Extra
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return
EndIF
If N Elements(P LIST) NE 0 Then *(Self.List)[0] = P LIST
If N elements(S LIST) NE 0 Then *(Self.List)[1] = S LIST
If N elements(P label) NE 0 Then Self.LabelNames[0] = P Label[0]
If N_elements(S_Label) NE 0 Then Self.LabelsNames[1] = S_Label[0]
If N_Elements(UPDOWN) NE 0 Then Self.UPDOWN = Keyword_Set(UPDOWN)
If n_elements(Multiple) NE 0 Then Self.Multiple = KeyWord_Set(Multiple)
If n elements (No Duplicates) NE 0 Then $
 Self.No_Duplicates = KeyWord_Set(No_Duplicates)
If n_elements(Min_Length) NE 0 Then Self.ListLength[0] = Min_Length
If N elements(Max Length) NE 0 Then Self.ListLength[1] = Max length
Self->MGS_GUIObject::SetProperty, _Extra = Extra
END ;SetProperty
: INIT
```

```
FUNCTION HBB TwoList::Init, $
P list=P List, S list = S List, $
P_Label = P_Label, S_Label = S_Label, $
UPDOWN = UPDOWN, Multiple = Multiple, $
Min_Length = Min_length, Max_length = Max_Length, $
No_Duplicates = No_Duplicates, $
Ref Extra = Extra
IF NOT Self->MGS GUIobject::INIT( Extra = Extra) Then Return, 0
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
self.list = PtrArr(2,/Allocate)
If n_elements(P_List) EQ 0 then $
p_list = ['Martin', 'Goofy', 'David', 'DubYa', 'Mark', 'Mickey',$
 'Craig', 'Minnie', 'JD', 'Henry', 'Pavel']
*Self.List[0] = P List
If n_elements(S_list) EQ 0 then $
*Self.List[1] = ["] Else *Self.list[1] = S_list
If n_elements(P_label) NE 0 then Self.LabelNames[0] = P_Label[0]
If n elements(S label) NE 0 Then Self.LabelNames[1] = S Label[0]
Self.UPDOWN = keyword set(UPDOWN)
Self.Multiple = keyword set(Multiple)
Self.No Duplicates = Keyword Set(No Duplicates)
If n_elements(Min_length) EQ 0 then min_L = 10 Else min_L = Min_Length[0]
If n_elements(Max_Length) EQ 0 Then max_L = 30 Else Max_L = Max_Length[0]
Self.ListLength = [min L, max L]
Return, 1
END; INIT
: CleanUp
PRO HBB_TwoList::CleanUp
Ptr Free, Self.List
```

```
END ;CleanUp
Defintion
PRO HBB_TwoList__Define
struct = {HBB TwoList, $
 INHERITS MGS_GUIOBJECT, $
 LabelNames:strArr(2), $;Labels for each list
 LabelID:LonArr(2), $;labelIDs
List:PtrArr(2), $ ;list contents
 ListID: LonArr(2), $; widgetIDs for lists
                ;1 = allow up/down movement
 UPDOWN:0, $
 Multiple:0,$ ;1 = allow multiple selections
 No_Duplicates: 0, $ ;are duplicates permitted in s_list?
 ListLength:IntArr(2)}; minmax list length (10, 30 default)
END
;-----END HERE-----
Ben Tupper
Bigelow Laboratory for Ocean Sciences
180 McKown Point Rd.
W. Boothbay Harbor, ME 04575
btupper@bigelow.org
;+
; NAME:
; HBB_TwoList
 PURPOSE:
The purpose of this widget/object to provide the user an interactive means of
 of adjusting the contents of a list. Items from a 'primary' list may be copied to a
;'secondary' list. Items in the secondary list may be removed or 'moved up' or
'moved down' relative to each other.
 REQUIREMENTS:
 INHERITS MGS_GUIObject
 CATEGORY:
 Widgets.
: CALLING SEQUENCE:
```

Self->MGS\_GUIObject::Cleanup

```
; objref = OBJ_NEW('HBB_TwoList')
ARGUMENTS:
None required
INITIALIZATION KEYWORDS: (see MGS_GUlobject)
P LIST (Get/Set) An vector of values for the primary list.
 The datatype is maintained as provided, but the List_Widget
 requires that the values be string. So most numeric types
 are acceptable when converted to STRING. The items in the
 the secondary list are selected from this list. If not provided,
 a dummy list (list of dummies?) is provided automatically.
S_LIST (Get/Set) A vector of items for the secondary list. Default = ["]
P_LABEL (Get/Set) A label for the primary list. Default = "
S_LABEL (Get/Set) A label for the secondary list. Default = "
MAX_LENGTH (Get/Set) The maximum number of rows shown in the list, Default = 30
MIN LENGTH (Get/Set) The minimum number of rows shown in the list. Default = 10
UPDOWN (Get/Set) Set this keyword to permit up/down shuffling of items in
the secondary list using the MoveUp/MoveDown buttons.
MULTIPLE (Get/Set) Set this keyword to permit multiple-item selection
 in either list.
 NO DUPLICATES (Get/Set) Set this keyword to prevent duplicate
 items from the primary list from appearing in the secondary list.
EXAMPLE:
o = Obj_New('HBB_TWOLIST',/UPDOWN)
o->GUI
 ;fiddle with the lists... over items left/right/up/down.
close the dialog
o->GUI
 ;note your earlier chioces are preserved
Obj_Destroy, o
 ;cleanup
MODIFICATION HISTORY:
written 3 APR 2001 Ben Tupper
Bigelow Laboratory for Ocean Science
btupper@bigelow.org
 Based upon PRC DualList widget.
4 APR 2001 Fixed bug when moving the top item DOWN. BT
 Changed the keyword SHUFFLE to UPDOWN for clarity.
9 APR 2001 Changed 'Add' to 'Copy' and placed UpDown buttons under the
 secondary list. Forced buttons to same size. BT
```

```
; Copyltem
FUNCTION HBB_TwoList::Copyltem, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
p_select = widget_info(self.listid[0],/List_select)
If p_select[0] NE -1 then begin
 ;if duplicates are not permitted, then
 ;check for duplication otherwise skip this step
If Self.No Duplicates EQ 1 Then Begin
 A = where(*Self.list[1] EQ (*self.list[0])[p_select], count)
 If Count NE 0 then Return, 0
 EndIf
if n_elements(*self.List[1]) GT 1 Then Begin
 *self.List[1] = [*self.list[1], (*self.List[0])[p_select]]
 EndIf Else begin
 if (*self.list[1])[0] EQ " Then $
 *self.List[1] = (*self.List[0])[p_select] Else $
 *self.List[1] = [*self.list[1], (*self.List[0])[p_select] ]
 EndELSE ;the secondary list is empty
Widget_control, Self.ListID[1], Set_Value = String(*Self.List[1])
EndIF; the p_select[0] ne -1
Return,0
END ;Copyltem event
; Removeltem
FUNCTION HBB_TwoList::RemoveItem, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
```

```
; if the secondary list is empty, then return immediately
If n_elements(*(Self.List)[1]) EQ 0 then Return,0
s_select = widget_info(self.listID[1], /List_select)
If s_select[0] NE -1 Then Begin
OldList = *(Self.List)[1]
If n_elements(OldList) GT 1 Then Begin
 index = replicate(1L, n elements(OldList))
 Index[s\_select] = 0
 A = where(index EQ 1, count)
 if Count gt 0 then *(Self.List)[1] = oldList[a] Else $
 *(self.List)[1] = "
 EndIf Else begin
 *(self.List)[1] = "
 EndElse; OldList only had one item to remove
Widget_control, Self.ListID[1], Set_Value = STRING(*(Self.List)[1])
EndIf; s select[0] ne -1
Return, 0
END:RemoveItem event
; MoveUp
FUNCTION HBB TwoList::MoveUp, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
s = widget_info(Self.ListID[1], /List_select)
If s[0] EQ -1 Then Return,0
;return if the list is 'empty'
if (*(Self.List)[1])[s[0]] EQ " Then Return,0
nsel = n_elements(s)
nList = n_elements(*(Self.List)[1])
;can't allow a higher index than elements in s_list
if s[nsel-1] GT (nlist-1) Then Return,0
```

```
if s[0] EQ 0 Then Begin ;move the top to the bottom
*(self.list)[1] = Shift(*(self.list)[1], -1)
EndIf Else begin
Index = Lindgen(Nlist)
Index[(s[0]-1L) : (s[nSel-1])] = SHIFT(Index[(s[0]-1L) : (s[nSel-1])], -1)
(*(Self.List)[1])= (*(Self.List)[1])[Index]
EndElse
Widget_control, Self.ListID[1], Set_Value = STRING( *(self.List)[1] )
Return, 0
END ; Move Up event
; MoveDown
FUNCTION HBB TwoList::MoveDown, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch. /Cancel
Return,0
EndIF
s = widget_info(Self.ListID[1], /List_select)
If s[0] EQ -1 Then Return,0
;return if the list is 'empty'
if (*(Self.List)[1])[s[0]] EQ " Then Return,0
nsel = n_elements(s)
nList = n_elements(*(Self.List)[1])
;can't allow a higher index than elements in s_list
if s[nsel-1] GT (nlist-1) Then Return,0
If s[nsel-1] EQ nList-1 then Begin
*(self.list)[1] = Shift(*(self.list)[1], 1) ;move the bottom to the top
EndIf Else begin
Index = Lindgen(Nlist)
If nsel Eq 1 then $
 Index[s[0]:s[0]+1L] = Shift(Index[s[0]:s[0]+1L],1) Else $
 Index[(s[0]-1L):(s[nSel-1]+1L)] = SHIFT(Index[(s[0]-1L):(s[nSel-1]+1L)],1)
(*(Self.List)[1])= (*(Self.List)[1])[Index]
EndElse
```

```
Widget_control, Self.ListID[1], Set_Value = STRING( *(self.List)[1] )
Return, 0
END ; Move Down event
; ListSelect - doesn't do anything (yet) except receive events
FUNCTION HBB_TwoList::ListSelect, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch. /Cancel
Return,0
EndIF
Return, 1
END :ListSelect
: BuildGUI
PRO HBB_TwoList::BuildGUI
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return
EndIF
colbase = widget base(self.layoutid, /base align center, column = 3)
width = (MAX(STRLEN(*self.list[0]))+5)>20
Height = self.ListLength[0] > (n_elements(*(Self.List)[0])+3) < Self.ListLength[1]
;column base for Primary List
PID = widget base(colbase, /base align center, column = 1, frame = 1)
pLabel = widget_Label(PID, Value = self.LabelNames[0], /align_center)
pList = Widget List(PID, Value = STRING(*(Self.List)[0]), $
ySize = height,xsize = width, $
Multiple = Self.Multiple, $
uValue = {Object:Self, Method: 'ListSelect' })
list_geo = widget_info(pList, /Geometry)
ButtonBase = Widget_Base(colbase, /base_align_center, column = 1, space = 10)
LabelID = Widget_Label(ButtonBase, value = 'Modify '+self.labelnames[1]+ 'Field', $
/align center)
```

```
Move = '>>> Move >>>'
Copy = '>>> Copy >>>'
Remove = '<< Remove <<'
MoveUp = ' Move Up '
MoveDown = ' Move Down '
TransferID = Widget Base(ButtonBase, Column = 1, /base align center)
AddID = Widget_Button(TransferID, Value = copy,$
 uValue = {Object:self, Method:'CopyItem'})
But geo = widget info(AddID, /Geometry)
RemoveID= Widget_Button(TransferID, Value = remove,$
 xsize = but geo.xsize, $
 uValue = {Object:self, Method:'RemoveItem'})
SID = widget Base(ColBase, /Base align center, Column = 1, Frame = 1)
sLabel = widget Label(SID, Value = self.LabelNames[1], /align center)
create the list with or without a value
if n_elements(*(self.list)[1]) NE 0 then $
sList = Widget_List(SID, Value = STRING(*(Self.List)[1]), $
 ysize = list_geo.scr_ysize, xsize = list_geo.scr_xsize, $
 Multiple = Self.Multiple, $
 uValue = {Object:Self, Method: 'ListSelect'}) Else $
sList = Widget List(SID, $
 ysize = list geo.scr ysize, xsize = list geo.scr xsize, $
 Multiple = Self.Multiple, $
 uValue = {Object:Self, Method: 'ListSelect' })
Self.ListID = [pList, sList]
UpID = Widget_Button(SID, Value = moveup,$
 xsize = but geo.xsize, $
 uValue = {Object:self, Method:'MoveUp' }, $
 sensitive = self.UPDOWN)
DownID = Widget Button(SID, value = movedown,$
 xsize = but geo.xsize, $
 uValue = {Object:self, Method:'MoveDown' }, $
 sensitive = self.UPDOWN)
END; BuildGUI
;----
```

```
; GetValue
FUNCTION HBB_TwoList::GetValue, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
Value = {S_LIST: *Self.List[1]}
Return, Value
END ;GetValue
; GetState
FUNCTION HBB_TwoList::GetState, event
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
If Widget Info(self.ListID[1], /Valid ID) Then $
S = Widget Info(self.ListID[1], /List Select) Else $
S = -1
State = {S_LIST:*Self.List[1], S_Select:S}
Return, State
END ;GetState
; GetProperty
PRO HBB_TwoList::GetProperty, P_list=P_List, S_list = S_List, $
P_Label = P_Label, S_Label = S_Label, $
UPDOWN = UPDOWN, Multiple = Multiple, $
Min_Length = Min_length, Max_length = Max_Length, $
_Ref_Extra = Extra
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
```

```
If arg_present(p_list) then P_List = *(Self.List)[0]
If Arg_present(S_LIST) Then s_List = *(Self.List)[1]
P_Label = Self.LabelNames[0]
S label = Self.LabelsNames[1]
UPDOWN = Self.UPDOWN
Multiple = Self.Multiple
No_Duplicates = Self.No_Duplicates
Min_Length = Self.ListLength[0]
Max_length = Self.ListLength[1]
Self->MGS GUIObject::GetProperty, Extra = Extra
END; SetProperty
SetProperty
PRO HBB_TwoList::SetProperty, P_list=P_List, S_list = S_List, $
P_Label = P_Label, S_Label = S_Label, $
UPDOWN = UPDOWN, Multiple = Multiple, $
Min Length = Min length, Max length = Max Length, $
No Duplicates = No Duplicates, $
Ref Extra = Extra
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return
EndIF
If N Elements(P LIST) NE 0 Then *(Self.List)[0] = P LIST
If N elements(S LIST) NE 0 Then *(Self.List)[1] = S LIST
If N_elements(P_label) NE 0 Then Self.LabelNames[0] = P_Label[0]
If N_elements(S_Label) NE 0 Then Self.LabelsNames[1] = S_Label[0]
If N_Elements(UPDOWN) NE 0 Then Self.UPDOWN = Keyword_Set(UPDOWN)
If n elements(Multiple) NE 0 Then Self.Multiple = KeyWord Set(Multiple)
If n elements(No Duplicates) NE 0 Then $
```

```
If n_elements(Min_Length) NE 0 Then Self.ListLength[0] = Min_Length
If N_elements(Max_Length) NE 0 Then Self.ListLength[1] = Max_length
Self->MGS_GUIObject::SetProperty, _Extra = Extra
END ;SetProperty
: INIT
FUNCTION HBB_TwoList::Init, $
P_list=P_List, S_list = S_List, $
P_Label = P_Label, S_Label = S_Label, $
UPDOWN = UPDOWN, Multiple = Multiple, $
Min Length = Min length, Max length = Max Length, $
No_Duplicates = No_Duplicates, $
Ref Extra = Extra
IF NOT Self->MGS GUIobject::INIT( Extra = Extra) Then Return, 0
Catch, Error
If Error NE 0 then Begin
Self->ErrorMessage
Catch, /Cancel
Return,0
EndIF
self.list = PtrArr(2,/Allocate)
If n elements(P List) EQ 0 then $
p_list = ['Martin', 'Goofy', 'David', 'DubYa', 'Mark', 'Mickey',$
 'Craig', 'Minnie', 'JD', 'Henry', 'Pavel']
*Self.List[0] = P_List
If n elements(S list) EQ 0 then $
*Self.List[1] = ["] Else *Self.list[1] = S_list
If n_elements(P_label) NE 0 then Self.LabelNames[0] = P_Label[0]
If n elements(S label) NE 0 Then Self.LabelNames[1] = S Label[0]
Self.UPDOWN = keyword set(UPDOWN)
Self.Multiple = keyword set(Multiple)
Self.No_Duplicates = Keyword_Set(No_Duplicates)
If n_elements(Min_length) EQ 0 then min_L = 10 Else min_L = Min_Length[0]
If n elements (Max Length) EQ 0 Then max L = 30 Else Max L = Max Length [0]
Self.ListLength = [min L, max L]
```

Self.No Duplicates = KeyWord Set(No Duplicates)

```
Return, 1
END; INIT
: CleanUp
PRO HBB TwoList::CleanUp
Ptr_Free, Self.List
Self->MGS_GUIObject::Cleanup
END ;CleanUp
: Defintion
PRO HBB TwoList Define
struct = {HBB TwoList, $
 INHERITS MGS_GUIOBJECT, $
 LabelNames:strArr(2), $;Labels for each list
 LabelID:LonArr(2), $;labelIDs
 List:PtrArr(2), $ ;list contents
 ListID: LonArr(2), $; widgetIDs for lists
 UPDOWN:0, $ ;1 = allow up/down movement
 Multiple:0,$ ;1 = allow multiple selections
 No_Duplicates: 0, $ ;are duplicates permitted in s_list?
 ListLength:IntArr(2)}; minmax list length (10, 30 default)
FND
File Attachments
1) hbb twolist define.pro, downloaded 112 times
```

Subject: Re: Object epiphany: A new way of building widget applications Posted by m.hadfield on Tue, 12 Jun 2001 02:50:50 GMT View Forum Message <> Reply to Message

From: "Martin Schultz" <martin.schultz@dkrz.de>

Hi Martin (and idl-pvwave groupies)

I finally got around to looking at your object widget stuff and I must say I am impressed (though I haven't yet grasped all the ins and outs). I like the

idea of storing an "object reference, method name" structure in each widget's UVALUE so that events from that widget can then be translated into method calls. Now why didn't I think of that?

---

Mark Hadfield m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield National Institute for Water and Atmospheric Research

--

Posted from clam.niwa.cri.nz [202.36.29.1] via Mailgate.ORG Server - http://www.Mailgate.ORG

Subject: Re: Object epiphany: A new way of building widget applications Posted by Martin Schultz on Tue, 12 Jun 2001 08:54:44 GMT View Forum Message <> Reply to Message

m.hadfield@niwa.cri.nz (Mark Hadfield) writes:

> From: "Martin Schultz" <martin.schultz@dkrz.de>

>

> Hi Martin (and idl-pvwave groupies)

>

- > I finally got around to looking at your object widget stuff and I must say I
- > am impressed (though I haven't yet grasped all the ins and outs). I like the
- > idea of storing an "object reference, method name" structure in each
- > widget's UVALUE so that events from that widget can then be translated into
- > method calls. Now why didn't I think of that?

>

Well, credit for this goes to David!! I merely carried his idea to the extreme ;-)

Cheers,

Martin

--

Subject: Re: Object epiphany: A new way of building widget applications Posted by John-David T. Smith on Tue, 12 Jun 2001 16:18:35 GMT

View Forum Message <> Reply to Message

## Mark Hadfield wrote:

>

> From: "Martin Schultz" <martin.schultz@dkrz.de>

>

> Hi Martin (and idl-pvwave groupies)

>

- > I finally got around to looking at your object widget stuff and I must say I
- > am impressed (though I haven't yet grasped all the ins and outs). I like the
- > idea of storing an "object reference, method name" structure in each
- > widget's UVALUE so that events from that widget can then be translated into
- > method calls. Now why didn't I think of that?

Hmm... I had used something like this a few years ago, but instead I just saved the "action" in the UVALUE, and in the event handling method I used something like:

widget\_control, ev.id, get\_uvalue=action if action eq 'bgroup' then action=ev.value case action of

'someaction': self->dosomething, /SOMETHING

....

else: call\_method,action,self ;all others, just call the method endcase

This is actually fairly nice, since you can add new functionality to the widget program without ever visiting the event code, but you don't \*require\* a method for all trivial events which occur, instead just catching those which aren't specifically handled and sending them on their way to special-purpose methods. Also note how I translate "actions" on the fly... so for instance a button and a menu item could trivially perform the same function. It also pares down the even handler in size... always a good thing.

JD

Subject: Re: Object epiphany: A new way of building widget applications Posted by Martin Schultz on Wed, 13 Jun 2001 09:42:53 GMT View Forum Message <> Reply to Message

JD Smith <idsmith@astro.cornell.edu> writes:

> Mark Hadfield wrote:

>>

```
>> From: "Martin Schultz" <martin.schultz@dkrz.de>
>>
>> Hi Martin (and idl-pvwave groupies)
>> I finally got around to looking at your object widget stuff and I must say I
>> am impressed (though I haven't yet grasped all the ins and outs). I like the
>> idea of storing an "object reference, method name" structure in each
>> widget's UVALUE so that events from that widget can then be translated into
>> method calls. Now why didn't I think of that?
>
> Hmm... I had used something like this a few years ago, but instead I
> just saved the "action" in the UVALUE, and in the event handling method
 I used something like:
>
> widget_control, ev.id, get_uvalue=action
> if action eq 'bgroup' then action=ev.value
  case action of
      'someaction': self->dosomething, /SOMETHING
>
>
      else: call_method,action,self ;all others, just call the method
  endcase
  This is actually fairly nice, since you can add new functionality to the
 widget program without ever visiting the event code, but you don't
> *require* a method for all trivial events which occur, instead just
> catching those which aren't specifically handled and sending them on
> their way to special-purpose methods. Also note how I translate
> "actions" on the fly... so for instance a button and a menu item could
> trivially perform the same function. It also pares down the even
> handler in size... always a good thing.
> JD
It appears to be the same idea behind this. My MGS_BaseGUI object provides
an event handler which is appropriate for many things and needs not to be
touched. So far the objref field in the widget UValue always contains
"self", and I admit, I am not sure why it should be something else ;-)
Cheers,
Martin
[[ Dr. Martin Schultz Max-Planck-Institut fuer Meteorologie
              Bundesstr. 55, 20146 Hamburg
[[
                                                     [[
              phone: +49 40 41173-308
[]
                                                   [[
```

[]

fax: +49 40 41173-298

 $\prod$ 

Subject: Re: Object epiphany: A new way of building widget applications Posted by m.hadfield on Wed, 13 Jun 2001 21:02:13 GMT

View Forum Message <> Reply to Message

From: "Martin Schultz" <martin.schultz@dkrz.de>

- > ... So far the objref field in the widget UValue always contains
- > "self", and I admit, I am not sure why it should be something else ;-)

Yeah, I was wondering about that :-)

---

Mark Hadfield m.hadfield@niwa.cri.nz http://katipo.niwa.cri.nz/~hadfield National Institute for Water and Atmospheric Research

--

Posted from clam.niwa.cri.nz [202.36.29.1] via Mailgate.ORG Server - http://www.Mailgate.ORG