Subject: Re: new graphics error message [_IDLITCONTAINER::SETPROPERTY]
Posted by bjkuk on Fri, 24 Feb 2012 02:49:40 GMT
View Forum Message <> Reply to Message

> bjkuk writes:
>> I did "Full_reset" and "help " command shows below.
>> Regards
>
>> IDL>
>> IDL> .Full_Reset
>> IDL> x = findgen(100)
>> IDL> y = sqrt(x)
>> IDL> pl = plot(x, y)
>> % Keyword PROXY not allowed in call to: _IDLITCONTAINER::SETPROPERTY
>> % Execution halted at: $MAIN$
>> IDL> help, /source
>> Compiled Procedures:
>> $MAIN$
>> Compiled Functions:
>> GRAPHICSWIN::GETPROPVALUE C:\Program Files\ITT\IDL\IDL81\lib
>> \graphics\graphicswin__define.pro
>> IDL>
>
> Well, now, this is VERY odd. :-)
>
> OK, let's try this. Let's open _IDLITCONTAINER up and
> *see* if it has a PROXY keyword on the the SetProperty
> method.
>
> IDL> .edit _IDLitContainer__Define
>
> This should open the file in the editor. Is it the right file?
> Located in C:\Program Files\ITT\IDL81\lib\itools\framework\?
> Find the SetProperty method. Do you see a PROXY keyword?
>
> Compile it. Try the code above again. What happens now?
>
> Cheers,
>
> David
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming:http://www.idlcoyote.com/

>

I can oopen exact file. and comple it. and code...
still same message like below.

and _IDLitContainer__Define.pro file is in exact directory. however i
cannot find any character "SetProperty" and "PROXY" in
_IDLitContainer__Define.pro
just in case, I have attached "_IDLitContainer__Define.pro"

Many thanks for your help.

Regards

```
IDL> .edit _IDLitContainer__Define
IDL> .compile -v 'C:\Program Files\ITT\IDL\IDL81\lib\itools\framework
\_idlitcontainer__define.pro'
% Compiled module: _IDLITCONTAINER::INIT.
% Compiled module: _IDLITCONTAINER::CLEANUP.
% Compiled module: _IDLITCONTAINER::GETPROPERTY.
% Compiled module: _IDLITCONTAINER::SETPROPERTY.
% Compiled module: _IDLITCONTAINER::_VALIDATEUNIQUEIDS.
% Compiled module: _IDLITCONTAINER::ADD.
% Compiled module: _IDLITCONTAINER::REMOVE.
% Compiled module: _IDLITCONTAINER::COUNT.
% Compiled module: _IDLITCONTAINER::GET.
% Compiled module: _IDLITCONTAINER::ISCONTAINED.
% Compiled module: _IDLITCONTAINER::MOVE.
% Compiled module: _IDLITCONTAINER::ADDBYIDENTIFIER.
% Compiled module: _IDLITCONTAINER::REMOVEBYIDENTIFIER.
% Compiled module: _IDLITCONTAINER::GETBYIDENTIFIER.
% Compiled module: _IDLITCONTAINER::_GETIDENTIFIERS.
% Compiled module: _IDLITCONTAINER::FINDIDENTIFIERS.
% Compiled module: _IDLITCONTAINER__DEFINE.
IDL> x=findgen(100)
IDL> y = sqrt(x)
IDL> pl = plot(x, y)
% Keyword PROXY not allowed in call to: _IDLITCONTAINER::SETPROPERTY
% Execution halted at: $MAIN$
IDL>
```

```
; $Id: //depot/idl/IDL_64/idldir/lib/itools/framework/
_idlitcontainer__define.pro#1 $
```

```
;
; Copyright (c) 2000-2007, ITT Visual Information Solutions. All
;       rights reserved. Unauthorized reproduction is prohibited.
 ;---------------------------------------------------------- ----------------
;+
; CLASS_NAME:
;   _IDLitContainer
;
; PURPOSE:
;   This file implements the _IDLitContainer class. This "abstract"
;   class provides functionality that allows container hierarchy
;   traversal using IDENTIFIERS.
;
;   This traversal is all built off of the identifier property
provided by
;   the IDLitComponent object.
;
;   Since this class is abstract, it doesn't provide the container or
;   component elemements needed for proper operation. Any class that
;   utilizes this functionality must subclass from IDL_Container and
;   the IDLitComponent object.
;
; CATEGORY:
;   IDL Tools
;
; SUPERCLASSES:
;   None.
;
; SUBCLASSES:
;
; CREATION:
;   See _IDLitContainer::Init
;
; METHODS:
;   This class has the following methods:
;
;   _IDLitContainer::Init
;   _IDLitContainer::Cleanup
;
; INTERFACES:
; IIDLProperty

;-
 ;---------------------------------------------------------- ----------------
; Lifecycle Routines
 ;---------------------------------------------------------- ----------------
; _IDLitContainer::Init
;
```

```
; Purpose:
; The constructor of the _IDLitContainer object.
;
; Parameter
;   None.
;
; Keyword:
;   CONTAINER   - The container to use for accessing children.

function _IDLitContainer::Init, _REF_EXTRA=_extra

  compile_opt idl2, hidden

   ; Set defaults.
   self._classname = 'IDL_Container'
   self._oChildren = self

   if (N_ELEMENTS(_extra) gt 0) then $
      self->_IDLitContainer::SetProperty, _EXTRA=_extra
   ; Does this class support the messaging interface? If so, add and
   ; removes will trigger messages
   self._bIsMessager = obj_isa(self, "IDLitIMessaging")
  return, 1
end
 ;--------------------------------------------------------- ----------------
; _IDLitContainer::Cleanup
;
; Purpose:
;    Destructor for the object.
;

pro _IDLitContainer::Cleanup

  compile_opt idl2, hidden

end


 ;--------------------------------------------------------- ----------------
; Properties
 ;--------------------------------------------------------- ----------------
; _IDLitContainer::GetProperty
;
; Purpose:
;   Used to get properties on this class
;
pro _IDLitContainer::GetProperty, $
   CLASSNAME=classname, $
```

```
    CONTAINER=container

    compile_opt idl2, hidden

    if ARG_PRESENT(classname) then $
        classname = self._classname

    if ARG_PRESENT(container) then $
        container = self._oChildren

end

 ;--------------------------------------------------------- ----------------
; Properties
 ;--------------------------------------------------------- ----------------
; _IDLitContainer::SetProperty
;
; Purpose:
;   Used to set properties on this class
;
; Keywords:
;  CLASSNAME   - Set to a string that is the classname of 'container'.
;  CONTAINER   - Set to the object that is 'container' for this
;                class.
pro _IDLitContainer::SetProperty, $
    CLASSNAME=classname, $
    CONTAINER=container


   compile_opt idl2, hidden

    if (SIZE(classname, /TYPE) eq 7) then $
        self._classname = classname

    if (N_ELEMENTS(container) eq 1) then begin
        if(not obj_isa(container, "IDL_Container"))then BEGIN
          Message, /CONTINUE, $

IDLitLangCatQuery('Message:Framework:ContInvalidClass')
          return
        endif

        self._oChildren = container
    endif

end

 ;--------------------------------------------------------- ----------------
```

```
; Implementation
 ;--------------------------------------------------------- ----------------
 ;--------------------------------------------------------- ----------------
;__IDLitContainer::_ValidateUniqueIDS
;
;Purpose:
;  Make sure that the items being added to the container have
;  unique identifiers. Any collision will cause the identifier
;  to be made "unique"
;
;  THIS IS ONLY FOR IDENTIFIERS AND SHOULD NOT MODIFY NAMES.
;
; Return Value:
;   1 - Success
;   0 - Error   - No IDs changed in this case.
;
; Parameters:
;   The canidates for addition.

function _IDLitContainer::_ValidateUniqueIDS, oNew

   compile_opt idl2, hidden

   ; get the objects in this container and grab
   ; their relative identifiers.
   oContained = self->_IDLitContainer::Get(/all, count=nObjs)
   nNew = n_elements(oNew)

    if(nObjs gt 0)then begin
        ; Verify no invalid objects were left in the container.
        bad = ~obj_valid(oContained)
        iNotValid = where(bad, nNotValid)
        if (nNotValid gt 0) then begin
            self->_IDLitContainer::Remove, oContained[iNotValid]
            nObjs = nObjs - nNotValid
            if (nObjs gt 0) then $
                oContained = oContained[WHERE(~bad)]
        endif
    endif

    if (~nObjs && nNew eq 1)then begin
        oNew[0]->IDLitComponent::GetProperty, IDENTIFIER=strID
        if (~strID) then $
            oNew[0]->IDLitComponent::SetProperty, IDENTIFIER='ID'
        return, 1
    endif

   if (nObjs gt 0) then begin
```

```
      ; Get the list of idents in this container.
      sIDs = strarr(nObjs)
      for i=0, nObjs-1 do begin
         oContained[i]->IDLitComponent::GetProperty,
IDENTIFIER=strID
         sIDs[i] = strID
      endfor
   endif else $
     sIDs = ''

   ; Now go through and check for unique Id
   for i=0, n_elements(oNew)-1 do begin

      ; If the object is already contained (or will be) then don't
      ; change its identifier.
      if ((nObjs && MAX(oContained eq oNew[i])) || $
         (i gt 0 && MAX(oNew[0:i-1] eq oNew[i]))) then $
         continue

      oNew[i]->IDLitComponent::GetProperty, IDENTIFIER=strID
      if (~strID) then $
         strID = 'ID'

      strNewID = IDLitGetUniqueName(sIDs, strID)
      if (strNewID ne strID) then $
         oNew[i]->IDLitComponent::SetProperty, IDENTIFIER=strNewID

      sIDs = [sIDs, strNewID] ; add new name to list of names
   endfor

   return, 1
end
 ;--------------------------------------------------------- ----------------
; _IDLitContainer::Add
;
; Purpose:
;   This method is similar to the Add method of an IDL container, but
;   it adds the given object to the child container.
;
; Parameters:
;   oObjects   - The item to add to the container.
;
; KEYWORDS:
;   NO_NOTIFY   - If set, don't send an notification message
;
;   USE__PARENT  - Use this objects _parent when setting _PARENT
;
;   All keywords are passed on to the final container, where the item
```

```
;   is added using the IDL_Container functionality.

pro _IDLitContainer::Add, oObjects, NO_NOTIFY=NO_NOTIFY, $
                USE__PARENT=USE__PARENT, _EXTRA=_EXTRA

    compile_opt idl2, hidden

    nItems = N_ELEMENTS(oObjects)
    if (~nItems) then $
        return

    ;Reject NULL objects, and reject myself.
    good = OBJ_VALID(oObjects) and oObjects ne self
    if (~ARRAY_EQUAL(good, 1b)) then begin
        iValid = WHERE(good, nItems)
        if (~nItems) then $
            return
        oTmp = oObjects
        oObjects = oObjects[iValid]
    endif


   if(keyword_set(USE__PARENT))then $
     self->IDLitComponent::Getproperty, _parent=parent $
   else $
    parent=self

   ; Set the logical parent for this component.
   for i=0, nItems-1 do $
    oObjects[i]->IDLitComponent::SetProperty, _PARENT=parent

   ; Okay, now verify that the ID names are unique
   iStatus = self->_IDLitContainer::_ValidateUniqueIDs( oObjects )


    if (KEYWORD_SET(no_notify)) then begin
        ; Assume we want to pass on NO_NOTIFY to our children,
        ; but bundle it up into _extra in case our child
        ; is just an IDL_Container.
        _extra = (N_TAGS(_extra) gt 0) ? $
            CREATE_STRUCT(_extra, 'NO_NOTIFY', 1) : {NO_NOTIFY: 1}
    endif

    CALL_METHOD, self._classname+'::Add', self._oChildren, $
        oObjects, _EXTRA=_extra

   ; Should a notify message be sent.
   if (self._bIsMessager && ~keyword_set(NO_NOTIFY)) then begin
```

```
      idItems = strarr(nItems)
      for i=0, nItems-1 do $
        idItems[i]=oObjects[i]->GetFullIdentifier()
      ; Notify that items were added
      self->IDLitiMessaging::DoOnNotify, parent->GetFullIdentifier(),
"ADDITEMS", idItems
  endif

   ; Restore my original object list.
   if (N_ELEMENTS(iValid) gt 0) then $
      oObjects = oTmp

end


;---------------------------------------------------------- ----------------
; _IDLitContainer::Remove
;
; Purpose:
;   Mimics the IDL_Container::Remove method, but takes into account
;   the logical child container that this object uses. Also verifies
;   that the object was removed and sends out notification.
;
; Parameters:
;   oObjects   - The item to remove from the container.
;
; KEYWORDS:
;   NO_NOTIFY   - If set, don't send an notification message
;
;   All keywords are passed on to the final container, where the item
;   is removed using the IDL_Container functionality.

pro _IDLitContainer::Remove, oObjects, $
   ALL=all, $
   NO_NOTIFY=NO_NOTIFY, $
   POSITION=position, $
   _EXTRA=_EXTRA

   compile_opt idl2, hidden

   ; If we don't have an input arg (or /ALL is set) then retrieve
   ; the objects we wish to remove. We need to do this here so
   ; we can do notification below.
   ; Note: according to the docs, POSITION should be ignored
   ; if we have an input arg. So we don't need to explicitly check
   ; for it in the if statement. The N_PARAMS will cover that case.
   if (N_PARAMS() eq 0 || KEYWORD_SET(all)) then $
      oObjects = self->_IDLitContainer::Get(ALL=all,
```

```
POSITION=position)

    nItems = N_ELEMENTS(oObjects)
    if (nItems eq 0) then $
        return

    CALL_METHOD, self._classname+'::Remove', self._oChildren, $
        oObjects, _EXTRA=_extra

    idItems = STRARR(nItems)
    oParents = OBJARR(nItems)

    ; First loop thru all removed objects to verify that they
    ; are still valid and that they were indeed removed.
    for i=0,nItems-1 do begin

        if (~OBJ_VALID(oObjects[i])) then $
            continue

        ; Make sure that object was actually removed from its parent.
        oObjects[i]->IDLitComponent::GetProperty, _PARENT=oParent
        ; Use _parent or myself.
        oParents[i] = OBJ_VALID(oParent) ? oParent : self

        ; If still contained, skip over this object.
        if (oParents[i]->IsContained(oObjects[i])) then $
            continue

        ; If removed, cache the object's full identifier.
        idItems[i] = oObjects[i]->GetFullIdentifier()
    endfor

    ; See how many objects actually got removed.
    iRemoved = WHERE(idItems, nRemoved)
    if (nRemoved eq 0) then $
        return

    ; Only keep the objects that were valid and were removed.
    oItems = oObjects[iRemoved]
    idItems = idItems[iRemoved]
    oParents = oParents[iRemoved]

    ; Should a notify message be sent?
    doNotify = self._bIsMessager && ~KEYWORD_SET(no_notify)

    for i=0, nRemoved-1 do begin
        if (doNotify) then $
            self->DoOnNotify, oParents[i]->GetFullIdentifier(), $
```

```
            "REMOVEITEMS", idItems[i]
        ; Null out parent.
        oItems[i]->IDLitComponent::SetProperty, _PARENT=OBJ_NEW()
    endfor

end


 ;--------------------------------------------------------- ----------------
function  _IDLitContainer::Count

   compile_opt idl2, hidden

    return, CALL_METHOD(self._classname+'::Count', self._oChildren)
end

 ;--------------------------------------------------------- ----------------
function  _IDLitContainer::Get, COUNT=COUNT, $
                      SKIP_PRIVATE=PRIVATE, _REF_EXTRA=_EXTRA

   compile_opt idl2, hidden

   oObjs = CALL_METHOD(self._classname+'::Get', self._oChildren, $
                     COUNT=COUNT, _EXTRA=_EXTRA)

   if (~count || ~keyword_set(private))then  $
    return, oObjs

   oOutObjs = oObjs
   nOut = 0
   isa = obj_isa(oObjs, "IDLitComponent")
   for i=0, count-1 do begin
      if(isa[i])then begin
         oObjs[i]->IDLitComponent::GetProperty, private=private
         if(not private)then $
           oOutObjs[nOut++] = oObjs[i]
      endif else $
        oOutObjs[nOut++] = oObjs[i]
   end
   count = nOut
   return, (nOut gt 0 ? oOutObjs[0:nOut-1] : -1)
end

 ;--------------------------------------------------------- ----------------
function  _IDLitContainer::IsContained, Object, _REF_EXTRA=_EXTRA

   compile_opt idl2, hidden
```

```
    return, CALL_METHOD(self._classname+'::IsContained',
self._oChildren, $
        Object, _EXTRA=_EXTRA)
end

 ;--------------------------------------------------------- ----------------
; _IDLitContainer::Move
;
; Purpose:
;    Used to override the move method so it can be directed to the
;    correct location.
;
; Parameters:
;    Source  - location of source
;
;    Destination - The new location to move to.
;
; Keywords
;   NO_NOTIFY   - If set, don't send an notification message
pro  _IDLitContainer::Move, Source, Destination, NO_NOTIFY=NO_NOTIFY

   compile_opt idl2, hidden

   oItem = self->Get(position=Source)

   CALL_METHOD, self._classname+'::Move', self._oChildren, $
     Source, Destination
   ; should we send a message?
   if(obj_valid(oItem) && self._bIsMessager and $
      ~keyword_set(NO_NOTIFY))then begin
       ; Should a notify message be sent?
       idItem = oItem->GetFullIdentifier()
       ; Notify that items were removed
       oItem->IDLitComponent::GetProperty, _PARENT=parent
       idParent = (obj_valid(parent) ? parent->GetFullIdentifier() : $
                   self->GetFullIdentifier())
       self->DoOnNotify, idParent, "MOVEITEMS", idItem
   endif
end


 ;--------------------------------------------------------- ----------------
; _IDLitContainer::AddByIdentifier
;
; Purpose:
;   This method emulates the Add method of the IDL container
;   class. This method will use the provided path to determine the
;   the actual container in the hierarchy to which the item should
```

```
;   be Added.
;
; Parameters:
; strID     - The ID to the location to add the item. If an empty
;             string, the value is added to this container.
;             Otherwise, the next entry in the path is
;             poped off, the contents are searched for a match of
;             that value and if a match is found, the search
;             continues.
;
; oAddee      - The item to be added to this hierarchy.
;
; Keywords:
;   FOLDER_CLASSNAME: By default, any subfolders within strID will be
;       automatically created if they don't exist. These subfolders
;       will be of the same class as the caller. Set the
FOLDER_CLASSNAME
;       keyword to a string giving the classname to be used when
;       creating the subfolders.
;
PRO _IDLitContainer::AddByIdentifier, strInput, oAddee, $
    FOLDER_CLASSNAME=folderClassname, $
    _EXTRA=_extra

   compile_opt idl2, hidden

   ; Absolute Path?
   if(strmid(strInput, 0, 1) eq "/")then begin
      ; This is an absolute Identifier. Determine if our identifier
      ; contains this object in this path. This way we can shortcut
      ; The process.
      idSelf = self->GetFullIdentifier()
      iPos = strpos(strupcase(strInput), idSelf)

      if (iPos eq -1) then begin ; no match, pass control to our
parent
         self->IDLitComponent::GetProperty, _PARENT=oParent
         if (obj_valid(oParent)) then begin
           oParent->AddByIdentifier, strInput, oAddee, _EXTRA=_extra
         endif else begin
           Message, $

 IDLitLangCatQuery('Message:Framework:InvalidDestinationId'), $
             /CONTINUE
         endelse
         return   ; we're done
      endif
```

```
      if (iPos gt 0) then begin
        Message, $
          IDLitLangCatQuery('Message:Framework:InvalidDestinationId'),
$
         /CONTINUE
        return
      endif

      strID = strmid(strInput, strlen(idSelf))  ; chop off the path to
this item.

   endif else strID = strInput

   strItem  = IDLitBasename(strID, remainder=strRemain, /reverse)
   if (strItem eq '') then begin
      ; Just add to this container!
      self->Add, oAddee, _EXTRA=_extra
      return
   endif

   oItems = self->Get(/ALL, COUNT=nItems)

   for i=0, nItems-1 do begin
      oItems[i]->IDLitComponent::GetProperty, IDENTIFIER=strTmp
      if (~STRCMP(strItem, strTmp, /FOLD_CASE)) then $
       continue
      ; If more information exists in the path and the
      ; object isa container traverse down
      if(strRemain eq '')then $
        oItems[i]->Add, oAddee, _EXTRA=_extra $
      else if( obj_isa(oItems[i], '_IDLitContainer'))then $
        oItems[i]->AddByIdentifier, strRemain, $
              oAddee, _EXTRA=_extra
      ; We're done.
      return
   endfor

    ; Didn't find the subfolder.
    ; Create it, add it to ourself, then add the item.
    class = (SIZE(folderClassname, /TYPE) eq 7) ? $
       folderClassname : OBJ_CLASS(self)
    oFolder = OBJ_NEW(class, NAME=strItem)
    self->Add, oFolder, _EXTRA=_extra
    oFolder->AddByIdentifier, strRemain, oAddee, _EXTRA=_extra

end
```

```
 ;---------------------------------------------------------- ----------------
; _IDLitContainer::RemoveByIdentifier
;
; Purpose:
;   This is similar to Remove, but the name is used to remove an
;   object  from the target container.
;
; Parameters:
;   strInput  - The ID of the object to remove from the hierarchy.
;
; Return Value:
;   The object removed from the system. No object found, a null
;   object reference is returned.

function _IDLitContainer::RemoveByIdentifier, strInput, $
    _EXTRA=_extra


   compile_opt idl2, hidden

   ; Absolute Path?
   if(strmid(strInput, 0, 1) eq "/")then begin
      ; This is an absolute Identifier. Determine if this identifier
      ; contains this object in this path. This way we can shortcut
      ; The process.
      idSelf = self->GetFullIdentifier()
      iPos = strpos(strupcase(strInput), idSelf)
      if(iPos eq -1)then begin ; no match, pass control to our parent
         self->IDLitComponent::GetProperty, _PARENT=oParent
         return, obj_valid(oParent) ? $
            oParent->RemoveByIdentifier(strInput, _EXTRA=_extra):
obj_new()
      endif else $
        ; chop off the path to this item.
        strID = strmid(strInput, strlen(idSelf))
  endif else strID = strInput

   strItem = IDLitBasename(strID, remainder=strRemain, /reverse)
   if(strItem eq '')then begin
      Message, "Error parsing identifier", /continue
      return, obj_new()
   endif
   ; Find the item we are searching for in this container.
   oItems = self->Get(/ALL, COUNT=nItems)

   for i=0, nItems-1 do begin
      oItems[i]->IDLitComponent::GetProperty, IDENTIFIER=strTmp
      if(strcmp(strItem, strTmp, /fold_case) ne 0)then begin
```

```
          ; Ok, we have a match, traverse. Do we traverse or return
          ; this item? Depends on the remainder value
          if(strRemain eq "")then begin
             self->Remove, oItems[i], _EXTRA=_extra
             return, oItems[i]
          endif else if(obj_isa(oItems[i], "_IDLitContainer"))then $
            return, $
             oItems[i]-
> _IDLitContainer::RemoveByIdentifier(strRemain, $
                 _EXTRA=_extra)
          break;
       endif
   endfor
   return, obj_new()
end

 ;--------------------------------------------------------- ----------------
; _IDLitContainer::GetByIdentifier
;
; Purpose:
;   This method emulates the Get method of the IDL container
;   class. This method will use the provided path to determine the
;   the actuall container in the hierachy from where the item should
;   be removed.
;
; Parameters:
; strInput    - The ID to the location to add the item. If an empty
;               string, the value is removed to this container.
;               Otherwise, the next entry in the path is
;               poped off, the contents are searched for a match of
;               that value and if a match is found, the search
;               continues.
;
; Return Value
;   The object being searched for. If nothing was found, a null
;   object is returned.

function _IDLitContainer::GetByIdentifier, strInput

   compile_opt idl2, hidden

    strInTmp = STRUPCASE(strInput[0])
    if (~strInTmp) then $
        return, obj_new()

   ; Absolute Path?
   if (STRCMP(strInTmp, '/', 1)) then begin
```

```
      ; This is an absolute Identifier. Determine if this identifier
      ; contains this object in this path. This way we can shortcut
      ; The process.
      idSelf = self->GetFullIdentifier()

      ; this is just this object
      if (strInTmp eq idSelf) then $
         return, self

      ; Shortcut to looking in the root directory.
      if (idSelf eq '/') then begin

        ; Strip off leading "/"
        strID = STRMID(strInTmp, 1)

      endif else begin

        ; Did we find a match at the beginning? We add a trailing
slash
        ; onto our own id to avoid accidental matches with ids that
start
        ; with the same chars, such as /TOOLS/PLOT_0 and /TOOLS/PLOT
        ;
        slen = STRLEN(idSelf)+1
        if (~STRCMP(strInTmp, idSelf+'/', slen)) then begin
            ; No match, pass control to our parent
           self->IDLitComponent::GetProperty, _PARENT=oParent
           return, obj_valid(oParent) ? $
              oParent->GetByIdentifier(strInTmp): obj_new()
        endif

        ; chop off the path to this item.
        strID = STRMID(strInTmp, slen)

      endelse

    endif else $
      strID = strInTmp

    strItem  = IDLitBasename(strID, remainder=strRemain, /reverse)
    if (~strItem) then begin
      Message, $
          IDLitLangCatQuery('Message:Framework:ErrorParsingIdentifier' )
+ $
         strID, /CONTINUE
      return, obj_new()
    endif
```

```
  oItems = self->Get(/ALL, COUNT=nItems)

  for i=0, nItems-1 do begin
    if (~obj_valid(oItems[i])) then $
      continue
    oItems[i]->IDLitComponent::GetProperty, IDENTIFIER=strTmp
    if (strItem eq strTmp) then begin
      ; If more information exists in the path and the
      ; object isa container traverse down
      if (~strRemain) then $
        return, oItems[i]
      if (obj_isa(oItems[i], "_IDLitContainer")) then $
        return, oItems[i]->GetByIdentifier(strRemain)
      break ; if we are here, this will case a null retval
    endif
  endfor

   ; Special case for DataSpaceRoot. We skipped over this from the
Layer,
   ; but certain items (such as manipulator undo/redo) may need to
retrieve
   ; the DataSpaceRoot using its identifier. This also works with
IDL60 code
   ; that may still have the "DATA SPACE ROOT" within a full
identifier.
   if (strItem eq 'DATA SPACE ROOT') then begin
     return, (strRemain eq '') ? self._oChildren : $
       self._oChildren->GetByIdentifier(strRemain)
   endif

  return, obj_new()
end


 ;--------------------------------------------------------- ----------
; Recursive part
;
pro _IDLitContainer::_GetIdentifiers, strArray, currString, leafNodes

   compile_opt idl2, hidden

   oItems = self->Get(/ALL, COUNT=nItems)

   for i=0, nItems-1 do begin

     oItems[i]->IDLitComponent::GetProperty, $
       IDENTIFIER=identifier, PRIVATE=private
```

```
        ; Skip if private.
        if (private) then $
            continue

        ; As we descend hierarchy, construct relative identifier.
        newString = currString + $
          (currString eq '' ? '' : '/') + identifier

        if (OBJ_ISA(oItems[i], '_IDLitContainer')) then begin

            oldcount = N_ELEMENTS(strArray) - 1

            ; If keeping all nodes, just append our container.
            if (~leafNodes) then $
                strArray = [strArray, newString]

            oItems[i]->_IDLitContainer::_GetIdentifiers, $
                strArray, newString, leafNodes

            ; If only keeping leaf nodes, then if our container
            ; had no (nonprivate) children, append it.
            if (leafNodes && (N_ELEMENTS(strArray)-1) eq oldcount)
then $
                strArray = [strArray, newString]

        endif else begin

            strArray = [strArray, newString]

        endelse

    endfor

end


 ;---------------------------------------------------------- ----------------
; Arguments:
;   Pattern: An optional argument giving the string pattern to match.
;       All identifiers within the container that match this pattern
;       (case insensitive) will be returned. If Pattern is not
supplied
;       then all identifiers within the container are returned.
;
; Keywords:
;   COUNT: Set this keyword to a named variable in which to return
;       the number of identifiers in Result.
;
```

```
;   LEAF_NODES: If this keyword is set then only leaf nodes will
;       be returned. The default is to return all identifiers that
;       match, including containers.
;
function _IDLitContainer::FindIdentifiers, Pattern, $
    COUNT=count, $
    LEAF_NODES=leafNodes

    compile_opt idl2, hidden

    ; Do recursive part.
    strArray = ''
    self->_IDLitContainer::_GetIdentifiers, strArray, '', $
        KEYWORD_SET(leafNodes)

    count = N_ELEMENTS(strArray) - 1  ; skip first null string
    if (count eq 0) then $
        return, ''

    strArray = strArray[1:*]

    ; Prepend my own identifier to construct full identifier.
    myID = self->GetFullIdentifier()

    ; Only add full identifier (begins with slash).
    if (STRMID(myID, 0, 1) eq '/') then begin
        ; Append a slash except for top-level node.
        if (myID ne '/') then $
            myID += '/'
        strArray = myID + strArray
    endif

    ; Filter returned strings.
    if (N_ELEMENTS(Pattern) ne 0) then begin
        matches = WHERE(STRMATCH(strArray, Pattern, /FOLD_CASE),
count)
        if (~count) then $
            return, ''
        ; Return string array or scalar string.
        return, (count gt 1) ? strArray[matches] :
strArray[matches[0]]
    endif

    ; Return string array or scalar string.
    return, (count gt 1) ? strArray : strArray[0]

end
```

```
 ;--------------------------------------------------------- ----------------
; Definition
 ;--------------------------------------------------------- ----------------
; _IDLitContainer__Define
;
; Purpose:
; Class definition of the object
;
pro _IDLitContainer__Define

   compile_opt idl2, hidden

   void = {_IDLitContainer,  $
          _bIsMessager: 0b, $      ; can perform notification and
should
          _classname : '', $       ; class of child container
          _oChildren : obj_new() } ;the actual child container
end
```