Subject: structures? pointers? Containers! (was: top10)
Posted by Martin Schultz on Fri, 11 Aug 2000 07:00:00 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Otte Homan (ottehoman@my-deja.com) writes:
>
>> I have the same problem - I'd like to create a
>> hierarchical data structure, basically an array,
>> of structures (so I can use DataFile(1),
>> DataFile(2), etc...) Each structure consists of
>> headers, parameters, and *dynamic* arrays. So the
>> structures are *more or less* the same, but
>> differ in the lenght of their arrays. Using IDL
>> 5.3 (sorry, our site has only this
>> version licensed) the contents of an array can
>> only be of one single type. My filestructures
>> are like this: {{header},{parameters},{data}},
>> where {data} is a structure with arrays of (from
>> file to file) different lengths. I only know at
>> runtime how long these arrays are.
>>
>> Any solution ?
>
> Well, I reiterate. Pointers. The solution is pointers. :-)
> [...]

I would vote for containers. They are even more flexible as you can store almost anything any size in there (as long as it is an object ;-) And the beauty of objects goes even further: they'll do everything automatically for you -- well, at least after you spent enough time and brain food to program them.

Cheers,
Martin

PS: I recollect postings from me saying "Nooo, I will never use objects!" Now I tend to write anything with them. BTW: thanks to Ben Tupper who pointed out a bug and a little misconception, I reiterated on my enhanced container object that allows object access by name (please find it attached). For IDL 5.3 or later you can now search names with the "real" Unix like pattern match (see StrMatch function). And the IsContained method now also accepts name(s) as input.
  Example: retrieve all objects whose name starts with B, D, or M and ends in n (sorry David!):
    theContainer->Get(Name='[BDM]*n')

--
[[[[[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[[[[[[

```
;+
; NAME:
;   MGS_Container (Object)
;
; PURPOSE:
;      This object extends the IDL_Container object (see online help)
;   by allowing access to objects by name. If objects stored in
;   the container possess a structure element called 'name' and allow
;   access to this element via a GetProperty method, MGS_Container
;   allows the following in excess of IDL_Container's functionality:
;   1. retrieve the container position objects via their name attributes
;   2. retrieve object references via the objects' name attributes
;   3. remove objects via their name attributes
;      Objects without a name attribute (or without a GetProperty
;   method) are internally handled as objects with an empty name string.
;      The Get, and Remove methods accept a single string or a string
;   vector with the 'name' keyword. If you run IDL >= 5.3, names may
;   contain Unix like wildcards ('*', '?', and '[...]'; see online
;   help for StrMatch), in IDL 5.2 they may at least end with a '*'.
;   A search expression '*' will return all objects including those
;   with no name attribute (or an empty string as name). To exclude
;   unnamed objects from the result, use the NoEmpty keyword.
;      A new method GetPosition accepts an object list as
;   argument. This can either be a list of object references or a
;   string (or string vector). This method returns an index vector
;   which can be used as 'position' value in other methods.
;
; AUTHOR:
;   Dr. Martin Schultz
;   Max-Planck-Institut fuer Meteorologie
;   Bundesstr. 55, D-20146 Hamburg
;   email: martin.schultz@dkrz.de
;
; CATEGORY:
;   Objects, General Programming
;
; CALLING SEQUENCE:
;   theContainer = Obj_New( 'MGS_Container' )
```

```
;   theContainer->Add, objectref
;   objectref = theContainer->Get( [keywords] )
;   Obj_Destroy, theContainer
;
; KEYWORDS TO THE INITIALIZATION ROUTINE:
;   None.
;
; PROCEDURES AND METHODS:
;   For a more detailed explanation on individual methods or
;   procedures, see the comments in the respective code sections.
;   The following list contains only "public" procedures and methods.
;   Methods marked with '*' are changed or new with respect to the
;   parent IDL_Container object.
;
; NON-OBJECT PROCEDURES:
;   None
;
; PROCEDURE METHODS:
;   MGS_Container::Add, objref [Position=index]
;           Add an object to the container.
;   MGS_Container::Move, source, dest
;           Move the position of an object within the container.
; * MGS_Container::Remove, [ objectref | Position=index | /All |
;                       Name=string ]
;           Delete an object from the container. NEW: can specify
;           objects by name.
;
; FUNCTION METHODS:
;   MGS_Container::Count()
;           Return number of objects stored in container.
; * MGS_Container::Get( /All | IsA=classname | Position=index |
;                     Name=string  [, Count=variable] )
;           Return references to selected objects from the
;           container. If no object matches the search criteria, a
;           long value of -1 is returned (as in IDL_Container).
;           NEW: can specify objects by name.
; * MGS_Container::IsContained( objref [, Position=index] )
;           Test if object(s) is stored in container.
; * MGS_Container::GetNames( [Position=index] )
;           NEW method. Return the names of the objects stored in the
;           container. Objects without a name field yield an empty string.
; * MGS_Container::GetPosition( objlist )
;           NEW method. Returns the index values of objects specified
;           by object reference or name. Objlist can be a string vector
;           where each string can have a trailing '*' as wildcard.
;
; MODIFICATION HISTORY:
;
```

```
;   mgs, 19 Jul 2000 : VERSION 1.00
;   mgs, 03 Aug 2000 : bug fix in Get - check for empty position
;                                 (thanks to Ben Tupper)
;                      IsContained method now also allows name
;                      specification and contains core of GetPosition
;                      method.
;   mgs, 10 Aug 2000 : now uses StrMatch for wildcard expansion if IDL>=5.3
;
;-
;
 ;############################################################## ##################
;
; LICENSE
;
; This software is OSI Certified Open Source Software.
; OSI Certified is a certification mark of the Open Source Initiative.
;
; Copyright ï¿½ 2000 Martin Schultz
;
; This software is provided "as-is", without any express or
; implied warranty. In no event will the authors be held liable
; for any damages arising from the use of this software.
;
; Permission is granted to anyone to use this software for any
; purpose, including commercial applications, and to alter it and
; redistribute it freely, subject to the following restrictions:
;
; 1. The origin of this software must not be misrepresented; you must
;    not claim you wrote the original software. If you use this software
;    in a product, an acknowledgment in the product documentation
;    would be appreciated, but is not required.
;
; 2. Altered source versions must be plainly marked as such, and must
;    not be misrepresented as being the original software.
;
; 3. This notice may not be removed or altered from any source distribution.
;
; For more information on Open Source Software, visit the Open Source
; web site: http://www.opensource.org.
;
 ;############################################################## ##################


;  ------------------------------------------------------------ ------------------------
; Undefine:
; This method applies a trick published by Andrew Coole, Adelaide, AUS
; to render an undefined variable from within an IDL subroutine.
```

```
pro MGS_Container::Undefine, arg

   if (n_elements(arg) eq 0) then return   ; already undefined
   tempvar = SIZE(TEMPORARY(arg))

end

 ;---------------------------------------------------------- ----------------
; GetOneName (*private*):
; This method returns the NAME field of the object stored at the
; position given as argument. If the object contains no NAME field or
; has no GetProperty method, an empty string is returned.
; Note: position must be a scalar!

FUNCTION MGS_Container::GetOneName, Position

   ;; Establish error handler in case object has no GetProperty method
   CATCH, theError
   IF theError NE 0 THEN BEGIN
      CATCH, /Cancel
      RETURN, ''
   ENDIF

   ;; Get the object reference of the specified object
   theObject = self->Get(Position=position)
   IF not Obj_Valid(theObject) THEN RETURN, ''

   ;; Atempt to call the object's GetProperty method to retrieve the
   ;; name
   theObject->GetProperty, name=name

   ;; If successful, return object name. Otherwise, error handler
   ;; should return empty string.
   RETURN, name

END


 ;---------------------------------------------------------- ----------------
; GetNames:
; This method returns the NAME fields of all objects stored in the
; container. If the optional POSITION keyword is used, only the names
; of the objects at the indices given by POSITION will be returned.
; Invalid position indices yield empty strings as names.

FUNCTION MGS_Container::GetNames, Position=position

   ;; If no position vector was passed, get number of objects stored
```

```
  ;; in container and create index list
  IF N_Elements(position) GT 0 THEN BEGIN
    index = position
  ENDIF ELSE BEGIN
    objcount = self->Count()
    IF objcount GT 0 THEN $
      index = lindgen(objcount) $
    ELSE $
      RETURN, ''
  ENDELSE

  ;; Create result array
  result = StrArr(N_Elements(index))

  ;; Get individual object names
  FOR i=0L, N_Elements(index)-1 DO $
    result[i] = self->GetOneName(index[i])

  RETURN, result

END


 ;------------------------------------------------------------ ----------------
; GetPosition:
; This method returns the position indices of the objects specified as
; object reference (in this case the IDL_Container method IsContained
; is used) or specified by names. A -1 is returned if no object in the
; container matches the search or if OBJLIST is neither of type string
; nor objref.
;   Object names can contain wildcards ('*', '?', and '[...]' for IDL
; >=5.3). Name search is case-insensitive unless you pass the
; case_sensitive keyword (see IsContained method). A search pattern
; '*' will also return unnamed objects unless you set the NoEmpty keyword.


FUNCTION MGS_Container::GetPosition, objlist,   $
                  case_sensitive=case_sensitive,  $
                  noempty=noempty


  ;; Establish error handler to be safe
  CATCH, theError
  IF theError NE 0 THEN BEGIN
    CATCH, /Cancel
    RETURN, -1L
  ENDIF
```

```
  ;; If no objlist is given, return immediately
  IF N_Elements(objlist) EQ 0 THEN RETURN, -1L

  ;; Check if objlist is of type object reference or string
  type = Size(objlist, /TNAME)

  ;; For these, use the inherited IsContained method
  IF type EQ 'OBJREF' OR type EQ 'STRING' THEN BEGIN
    dummy = self->IsContained(objlist,   $
                        case_sensitive=case_sensitive,  $
                        noempty=noempty,  $
                        Position=index )
    RETURN, index
  ENDIF

  ;; Reaching this portion of the routine means invalid type for
  ;; objlist. Simply return -1.
  Message, 'Objlist must be of type OBJ_REF or STRING!', /Continue
  RETURN, -1L
END


 ;------------------------------------------------------------ ----------------
; Get:
; This method returns object references for objects that match
; specified criteria - or all objects if the All keyword is set.
; The IDL_Container method Get is extended to include a Name
; keyword. All and Position take precedence over name. As in the
; original, count can be used to return the number of objects
; retrieved.

FUNCTION MGS_Container::Get, All=all,   $
                  Position=position, $
                  Name=name,  $
                  Case_Sensitive=case_sensitive,  $
                  NoEmpty=NoEmpty,  $
                  Count=count,  $
                  _Ref_Extra=e


  ;; Initialize count value
  count = 0L

  ;; If All or Position keywords are used, call inherited method
  IF Keyword_Set(all) THEN  $
    RETURN, self->IDL_Container::Get(/All, count=count)

  IF N_Elements(Position) GT 0 THEN $
```

```
    RETURN, self->IDL_Container::Get(Position=position, count=count)

  ;; Look for named objects ?
  IF N_Elements(name) GT 0 THEN BEGIN
    ;; Find object positions
    namepos = self->GetPosition(name, $
                          case_sensitive=case_sensitive,  $
                          noempty=noempty)
    ;; If no match, return empty object reference
    IF namepos[0] LT 0 THEN BEGIN
      dummy = obj_new()
      RETURN, dummy
    ENDIF
    ;; Return objects
   RETURN, self->IDL_Container::Get(Position=namepos, count=count)
  ENDIF

  ;; Otherwise call inherited method with extra keywords
  RETURN, self->IDL_Container::Get(_extra=e, count=count)

END


 ;--------------------------------------------------------- ----------------
; IsContained:
; This method tests if a specific object reference or an object with
; a certain name is in the container. The IsContained method of
; IDL_Container is extended so that it checks the type of objref.
; If ChildObject is a string or string array, a pattern match is
; performed. For IDL >= 5.3 the StrMatch function is used which allows
; Unix like wildcard search ('*', '?', and '[...]'). For IDL <= 5.2 a
; primitive wildcard search is implemented which only allows for a
; trailing '*'.


FUNCTION MGS_CONTAINER::IsContained, ChildObject,   $
                  Position = Position,  $
                  Case_Sensitive=case_sensitive, $
                  NoEmpty=NoEmpty

  ;; If ChildObject is empty then return immediately
  IF N_Elements(ChildObject) EQ 0 THEN RETURN,0

  type = Size(ChildObject, /TName)

  CASE Type OF
    'OBJREF':  BEGIN
      result =  Self->IDL_Container::IsContained(ChildObject, Position = Position)
```

```
  RETURN, result
END

'STRING':  BEGIN
  ;; For strings, interpret them as object names. Get the names of
  ;; all objects and apply (version dependent) pattern matching.
  result = 0
  position = -1L
  names = self->GetNames()
  names = StrTrim( names, 2)
  IF NOT Keyword_Set(case_sensitive) THEN $
    names = StrUpCase(names)

  ;; Return if container has no objects stored
  IF self->Count() EQ 0 THEN RETURN, result

  ;; Loop through objlist and find matching indices
  FOR i=0L, N_Elements(ChildObject)-1 DO BEGIN
    testname = StrTrim( ChildObject[i], 2)

    IF (!Version.Release GE 5.3) THEN BEGIN
      ;; Use the StrMatch function
      ;; StrMatch returns a logical array with 1 for every
      ;; string that matches the search expression
      lw =  StrMatch(names,testname,fold_case=1-Keyword_Set(case_sensiti ve))
      ;; Convert logical array to index array
      w = Where(lw, wcnt)
    ENDIF ELSE BEGIN
      ;; Restricted wildcard use (only '*' at the end of the expression)
      ;; Check if wildcard is used in name
      IF NOT Keyword_Set(case_sensitive) THEN $
        testname = StrUpCase(testname)
      sloppy = 0
      IF (( p = StrPos(testname,'*') )) GE 0 THEN BEGIN
        l = StrLen(testname)
        IF l-p NE 1 THEN BEGIN ; wildcard must be last character
          Message, 'Invalid name specification '+ChildObject[i]+'!', /Continue
          GOTO, skip_it
        ENDIF
        testname = StrMid(testname, 0, l-1)
        sloppy = 1
      ENDIF

      ;; Look for matching object names
      IF sloppy THEN BEGIN
        w = Where(StrPos(names,testname) EQ 0, wcnt)
      ENDIF ELSE BEGIN
        w = Where(names EQ testname, wcnt)
```

```
            ENDELSE
          ENDELSE

          ;; Add indices to list
          IF wcnt GT 0 THEN position = [ position, w ]

skip_it:
        ENDFOR

        ;; Remove first dummy position if any names have been found
        ;; and set result value to 1
        IF N_Elements(position) GT 1 THEN BEGIN
          result = 1
          position = position[1:*]
        ENDIF

        ;; Remove position of unnamed objects if NoEmpty keyword is
        ;; set
        IF Keyword_Set(NoEmpty) AND position[0] GE 0 THEN BEGIN
          we = Where(names EQ '', wcnt)
          IF wcnt GT 0 THEN BEGIN
            ;; Convert index array to logical array
            ;; and delete "empty" elements
            larr = intarr(N_Elements(names))
            larr[position] = 1
            larr[we] = 0
            position = Where(larr, wcnt)
            IF wcnt EQ 0 THEN result = 0
          ENDIF
        ENDIF

        RETURN, result
      END

      ELSE:      BEGIN
        Message, 'ChildObject must be of type OBJREF or STRING!',/Continue
        RETURN,0
      END
    ENDCASE
END


 ;--------------------------------------------------------- ----------------
; Remove:
; This method removes objects from the container. Similarily to the
; Get method, this method extends the IDL_Container::Remove method to
; allow object specification by name.
; Note: Unlike the original method of IDL_Container, the child_object
```

```
; argument, or the position or name keywords can contain more than one
; element. The unique position values are sorted and reversed before
; removing objects.

PRO MGS_Container::Remove, child_object,    $
              name=name,     $
              case_sensitive=case_sensitive,  $
              noempty=noempty,  $
              position=position, $
              All=all,   $
              _Ref_Extra=e

  ;; If all keyword is set, call parent method accordingly
  IF Keyword_Set(all) THEN BEGIN
    self->IDL_Container::Remove, /All
    RETURN
  ENDIF

  ;; If object references are given, loop over them and call the
  ;; parent object's remove method for each of them
  IF N_Elements(child_object) GT 0 THEN BEGIN
    FOR i=0L, N_Elements(child_object)-1 DO $
      self->IDL_Container::Remove, child_object[i]
    RETURN
  ENDIF

  ;; If names are provided, convert them to position indices
  IF N_Elements(name) GT 0 THEN $
    index = self->GetPosition(name, $
                        case_sensitive=case_sensitive,  $
                        noempty=noempty)

  ;; If position indices are provided, make local copy
  IF N_Elements(position) GT 0 THEN $
    index = long(position)

  ;; Handle name and position cases: find unique position values,
  ;; sort and reverse them and loop
  maxindex = self->Count()
  IF N_Elements(index) GT 0 THEN BEGIN
    ;; Need to sort index array and remove items from the end
    index = Reverse( index[ Uniq( index, Sort(index) ) ] )
    ok = where(index GE 0 AND index LT maxindex, cnt)
    IF cnt EQ 0 THEN RETURN    ;; no valid index values
    index = index[ok]
    FOR i=0L, N_Elements(index)-1 DO $
      self->IDL_Container::Remove, Position=index[i]
    RETURN
```

```
  ENDIF

  ;; Handle other (future extensions) cases
  self->IDL_Container::Remove, _extra=e

END


;--------------------------------------------------------- ----------------
; Cleanup:
; This method frees all object values and destroys all objects stored
; in the container. This is already handled in IDL_Container, so
; Cleanup only needs to call the inherited object's method.

PRO MGS_Container::Cleanup

  self->IDL_Container::Cleanup

END


;--------------------------------------------------------- ----------------
; Init:
; This method initializes the object values. There are no arguments,
; so the only action is to call the parent's Init method

FUNCTION MGS_Container::INIT

  RETURN, self->IDL_Container::Init()

END


;--------------------------------------------------------- ----------------
; MGS_Container__Define:
; This is the object definition for the container object. It inherits
; everything from IDL_Container.

PRO MGS_Container__Define

  object_class = {  MGS_Container,    $
                    INHERITS  IDL_Container  $
                 }

END
```

File Attachments

1) mgs_container__define.pro, downloaded 80 times