Subject: Re: LIST extensions
Posted by penteado on Tue, 14 Dec 2010 22:31:43 GMT
View Forum Message <> Reply to Message

On Dec 14, 7:51 pm, Paul van Delst <paul.vande...@noaa.gov> wrote:
> Hello,
>
> Sorry to keep flogging this particular topic, but I'm currently up to my eyes in adapting some code from using pointers
> to using lists (partly for curiosity, partly to avoid nested pointer dereferencing), and I've found the very simple
> methods below to be useful in dealing with the "is it empty?", "are there nulls in the list?" etc type of questions I've
> been asking here.
>
> Comments?

This is very pertinent. Those are important needed features / bug
fixes that were missed in the way lists and hashes were implemented.
They are good ideas to add to the class I started implementing, which
will also include deep copy, conversion to/from pointer arrays, and
fix those issues with non existing elements.

Subject: Re: LIST extensions
Posted by Matt Haffner on Wed, 15 Dec 2010 06:51:32 GMT
View Forum Message <> Reply to Message

On Dec 14, 3:51 pm, Paul van Delst <paul.vande...@noaa.gov> wrote:
> Hello,
>
> Sorry to keep flogging this particular topic, but I'm currently up to my eyes in adapting some code from using pointers
> to using lists (partly for curiosity, partly to avoid nested pointer dereferencing), and I've found the very simple
> methods below to be useful in dealing with the "is it empty?", "are there nulls in the list?" etc type of questions I've
> been asking here.
>
> Comments?
>
> cheers,
>
> paulv
>
> ;+
> ; NAME:
> ; List::Empty

```
>  ;
>  ; PURPOSE:
>  ;       The List::Empty function method return TRUE if a list contains
>  ;       no elements, FALSE otherwise.
>  ;
>  ; CALLING SEQUENCE:
>  ;       result = Obj->[List::]Empty()
>  ;-
>  FUNCTION List::Empty
>    RETURN, N_ELEMENTS(self) EQ 0
>  END
>
>  ;+
>  ; NAME:
>  ;       List::Length
>  ;
>  ; PURPOSE:
>  ;       The List::Length function method returns the number of
>  ;       elements in a list.
>  ;
>  ; CALLING SEQUENCE:
>  ;       result = Obj->[List::]Length()
>  ;-
>  FUNCTION List::Length
>    RETURN, N_ELEMENTS(self)
>  END
>
>  ;+
>  ; NAME:
>  ;       List::n_Items
>  ;
>  ; PURPOSE:
>  ;       The List::n_Items function method returns the number of
>  ;       non-null elements in a list.
>  ;
>  ; CALLING SEQUENCE:
>  ;       result = Obj->[List::]n_Items()
>  ;-
>  FUNCTION List::n_Items
>    idx = WHERE(self NE !NULL, count)
>    RETURN, count
>  END
>
>  ;+
>  ; NAME:
>  ;       List::Compact
>  ;
>  ; PURPOSE:
```

```
>  ;       The List::Compact function method returns a copy of a list
>  ;       with all null elements removed.
>  ;
>  ; CALLING SEQUENCE:
>  ;       result = Obj->[List::]Compact()
>  ;-
>  FUNCTION List::Compact
>    idx = WHERE(self NE !NULL, count)
>    RETURN, ( count GT 0 ) ? self[idx] : LIST()
>  END
```

Useful--thanks.

Also, note that LIST is a subclass of IDL_Container, which has
a .Count() method, so .Length() may not be needed. There is also
a .Move method to rearrange items in a container. Unfortunately
the .IsContained method doesn't seem to work for me on a LIST though
(and is only for objects, in any case):

```
IDL> z=g[50]
IDL> print,g.IsContained(z)
           0
IDL> print,obj_valid(z, /get_heap)
     1884104
IDL> print,obj_valid(g[50], /get_heap)
     1884104
IDL> print, z eq g[50]
   1
```

mh

Subject: Re: LIST extensions
Posted by Paul Van Delst[1] on Wed, 15 Dec 2010 14:54:51 GMT
View Forum Message <> Reply to Message

Matt Haffner wrote:
> Also, note that LIST is a subclass of IDL_Container, which has
> a .Count() method, so .Length() may not be needed.

Excellent tip. I did not know that LIST was a subclass of IDL_Container. I will be sure to check that sort of thing with
other IDL objects in the future, i.e.

```
IDL> l=list(1,2,34,4)
IDL> help, l, /object
** Object class LIST, 2 direct superclasses, 4 known methods
   Superclasses:
      IDL_CONTAINER  <Direct>
```

```
      IDL_OBJECT  <Direct>
   Known Function Methods:
      IDL_CONTAINER::COUNT
      LIST::INIT
      LIST::_OVERLOADHELP
   Known Procedure Methods:
      LIST::ADD
IDL> help, l.count()
<Expression>    LONG      =           4
```

And the Get method worked even though the list contains no objects:

```
IDL> help, l.get(position=2)
<PtrHeapVar4>   LONG      =          34
```

As does the Move:

```
IDL> print, l
           1
           2
          34
           4
IDL> l.move,2,0
IDL> print, l
          34
           1
           2
           4
```

Tres cool!

I don't know why I should be surprised. But it would be nice if the superclasses were listed in the documentation.

Again, thanks for the info.

cheers,

paulv

> There is also
> a .Move method to rearrange items in a container. Unfortunately
> the .IsContained method doesn't seem to work for me on a LIST though
> (and is only for objects, in any case):
>
> IDL> z=g[50]
> IDL> print,g.IsContained(z)
>   0

> IDL> print,obj_valid(z, /get_heap)
>   1884104
> IDL> print,obj_valid(g[50], /get_heap)
>   1884104
> IDL> print, z eq g[50]
>   1
>
> mh

Subject: Re: LIST extensions
Posted by penteado on Wed, 22 Dec 2010 23:47:37 GMT
View Forum Message <> Reply to Message

On Dec 14, 8:31 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> This is very pertinent. Those are important needed features / bug
> fixes that were missed in the way lists and hashes were implemented.
> They are good ideas to add to the class I started implementing, which
> will also include deep copy, conversion to/from pointer arrays, and
> fix those issues with non existing elements.

Other changes I am considering to put in my derived classes:

1) Make lists do nothing (as hashes already do) if !null is used as
index on _overloadBracketsLeftSide.

2) Make lists and hashes return !null when !null is used as an index
(now they throw an error).

3) Make lists and hashes accept !null on the _overloadPlus method and
do nothing, instead of throwing an error.

(3) is to work in conjunction with (2), so that lists/hashes can be
added to indexed lists/hashes, without having to verify if the index
is not !null.

Any thoughts?

Subject: Re: LIST extensions
Posted by penteado on Sun, 02 Jan 2011 06:37:49 GMT
View Forum Message <> Reply to Message

On Dec 22 2010, 9:47 pm, Paulo Penteado <pp.pente...@gmail.com> wrote:
> Other changes I am considering to put in my derived classes:
>
> 1) Make lists do nothing (as hashes already do) if !null is used as

> index on _overloadBracketsLeftSide.
>
> 2) Make lists and hashes return !null when !null is used as an index
> (now they throw an error).
>
> 3) Make lists and hashes accept !null on the _overloadPlus method and
> do nothing, instead of throwing an error.
>
> (3) is to work in conjunction with (2), so that lists/hashes can be
> added to indexed lists/hashes, without having to verify if the index
> is not !null.
>
> Any thoughts?

I have really been finding inconvenient the lack of these, and noticed another shortcoming: _overloadPlus should add to a list something that is not a list. So that

```
l1=list()
l2=list(1,2,3)
w=where(l2.toarray() eq 2)
l1+=l2[w]
```

Does not throw an error. As it is now, it takes a lot of work to select elements from a list with where(): not only it is necessary to test for no results (because !null is not accepted as index for lists), but it is also necessary to test for a single match, as a list indexed by a scalar (or 1-element array) returns the list element, which cannot be concatenated to a list (unless the element happens to be a list, which would not throw an error, but would concatenate in the wrong way).

An alternative is not change _overloadPlus, but change _overloadBracketsRightSide to return a 1-element list when given a 1-element array as index. It should still return the element when indexed by a scalar.

And doing these things also makes me think that, for syntatic sugar, there should be a list::where() method that would simply call where() on the list's toarray() result. Or where() should automatically call toarray() if given a list.
