Subject: Re: LIST extensions
Posted by penteado on Sun, 02 Jan 2011 07:14:06 GMT
View Forum Message <> Reply to Message

On Jan 2, 4:37 am, Paulo Penteado <pp.pente...@gmail.com> wrote:
> 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.

Also, the remove method should not remove elements when the index it
is given is !null. For the same reason, as, in the example above,

l2.remove,where(l2.toarray() lt 0,/null)

Will remove the last element of l2, instead of removing nothing.

But this has an implementation difficulty: how can a routine
distinguish between being given !null for an argument, and not being
given that argument? I remember this being asked, but do not remember
if there was an answer. The only way I can think of now is to try to
concatenate something to it, and catch the error that gets thrown in
case the variable is not !null. As in:

function pp_isnull,a

```
compile_opt idl2,logical_predicate
catch,err
if err then begin
  catch,/cancel
  ret=0
endif else begin
  b=[a,a]
  ret=1
endelse
return,ret
end
```