Subject: Storing !NULL in struct
Posted by tom.grydeland on Tue, 12 Mar 2013 12:09:37 GMT
View Forum Message <> Reply to Message

Any reason I shouldn't expect this to work?

IDL> help, { foo: !null }

--T

Subject: Re: Storing !NULL in struct
Posted by lecacheux.alain on Fri, 15 Mar 2013 13:31:18 GMT
View Forum Message <> Reply to Message

Le vendredi 15 mars 2013 09:10:45 UTC+1, Tom Grydeland a écrit :
>> On 3/13/13 2:17 am, Tom Grydeland wrote:
>
>>> And it would be useful _to me_ if I could put a !NULL in a struct.
>
>>> I'm using the struct for UVALUEs in widgets, and it would be nice if
>
>>> they allowed _any_ well-defined IDL value.
>
>
>
> On Wednesday, March 13, 2013 2:46:17 PM UTC, Mike Galloy wrote:
>
>> But !null is undefined:
>
>
>
> Not so undefined that I cannot assign it to a variable, or return it from a function, or use it as a good 'missing value' indicator in most cases. It is also pretty damn useful if you want to build an array by parts (although, as Bob pointed out, I could use a LIST for that last case).
>
>
>
> See, I was trying to stash away an arbitrary value (provided by the user for their later retrieval) under a given tag in a structure, and I didn't want to restrict what values they were allowed to use.
>
>
>
> So instead of using a struct { internal: <whatever>, public: <your value here> }, I thought I could use a List(<whatever>, <your value here>), but that didn't work either, since I'm not allowed to assign to a struct that lives in a List.
>
>

>
> IDL> c = List({ t: 0})
>
> IDL> print, c[0].t
>
>  0
>
> IDL> c[0].t = 1
>
> % Attempt to store into an expression: Structure reference.
>
> % Execution halted at: $MAIN$
>
>
>
> :-(
>
>
>
>> Mike
>
>
>
> --T

Are'nt you mixing three different concepts: undefined value, unassigned value and missing data ?
In IDL, you can use : !Null, New_Ptr() and !Values.F_NAN, respectively.

alx.

---

Subject: Re: Storing !NULL in struct
Posted by Lajos Foldy on Fri, 15 Mar 2013 14:02:19 GMT
View Forum Message <> Reply to Message

On Friday, March 15, 2013 9:10:45 AM UTC+1, Tom Grydeland wrote:
>
> IDL> c = List({ t: 0})
> IDL> print, c[0].t
>  0
> IDL> c[0].t = 1
> % Attempt to store into an expression: Structure reference.
> % Execution halted at: $MAIN$

This is a bug IMHO. This construct should work as a structure array works (LIST is a pointer array in disguise):

IDL> a=replicate({t:0}, 1)

```
IDL> a[0].t=1
IDL> print, a[0].t
       1
```

regards,
Lajos

---

Subject: Re: Storing !NULL in struct
Posted by kagoldberg on Fri, 15 Mar 2013 16:40:55 GMT
View Forum Message <> Reply to Message

It's not a bug (IMHO) because structures are rigid types after they are defined. If you use a pointer, you can have the same functionality you seek, and do anything you want with any type of data, after the structure is defined.

---

Subject: Re: Storing !NULL in struct
Posted by Michael Galloy on Fri, 15 Mar 2013 18:51:51 GMT
View Forum Message <> Reply to Message

On 3/15/13 2:10 AM, Tom Grydeland wrote:
> On Wednesday, March 13, 2013 2:46:17 PM UTC, Mike Galloy wrote:
>> But !null is undefined:
>
> Not so undefined that I cannot assign it to a variable, or return it
> from a function, or use it as a good 'missing value' indicator in
> most cases. It is also pretty damn useful if you want to build an
> array by parts (although, as Bob pointed out, I could use a LIST for
> that last case).

Nothing is so undefined that you can't return it from a function. !null is special in that you can assign it to a variable, so assigning it to a structure element would make some sense. It's just that I think the typical use case for a structure is not to rebuild the whole thing when you want to change a value (like you would if you switched the value from undefined type to any other type).

Why not a HASH? It's made for more dynamic situations like this.

Mike
--
Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)
Research Mathematician
Tech-X Corporation

---

Subject: Re: Storing !NULL in struct
Posted by chris_torrence@NOSPAM on Fri, 15 Mar 2013 20:40:56 GMT
View Forum Message <> Reply to Message

Hi all,

Not to stir the pot some more, but I actually looked into supporting !null in IDL structures. The deal-killer was that IDL structures are supposed to map directly to C structures. So you can theoretically take an IDL structure, write it out to a file, and then define a corresponding structure in C code and read it into that program. So having a !null field would break that compatibility.

Also, there were major issues of memory management. For named structures, if you allowed !null as a field, and then you decided to assign it a value, you would need to track down every instance of that named structure and re-define it. Even if you only allowed !null for anonymous structures, you could have an array of those structures. If you then defined the !null field in the first structure of the array to have some value, you would have to copy the entire array, and reallocate every structure to have the new value.

In short, it was just too much of a code overhaul and backwards compatibility risk to allow !null in structures.

As Mike suggests, perhaps HASH is the way to go.

Cheers,
Chris
ExelisVIS

Subject: Re: Storing !NULL in struct
Posted by penteado on Fri, 15 Mar 2013 23:14:14 GMT
View Forum Message <> Reply to Message

I think a lot about this subject.

The problem with doing

c[0].t=1

is in the way the overloaded brackets are implemented. When c[0] gets
the ".t", there is a function call to list's
_overloadBracketsRightSide, which just returns a value (the
structure). Since it is a value (not a variable), no values can be
assigned to it. To put this another way, the line above is the same as
trying to do

a=0
a+9=5

One way to sort of get around this would be to put in the list
pointers to the structures:

```
IDL> c=list(ptr_new({t:0}))
IDL> print,(*c[0]).t
       0
IDL> (*c[0]).t=9
IDL> print,(*c[0]).t
       9
```

Or one could make a derived list class that stored the elements by
pointers, and returned the pointers, so that one would not need to
keep doing ptr_new() everytime something is added to the list.

It could even return either the value or a pointer to it, depending on
how the brackets are used: If it gets an integer index, it returns the
element, as usual; if it gets a floating point index, it returns the
pointer to the element. Then it could be used like:

```
IDL> c=listbypointers({t:0})
IDL> print,c[0].t
       0
IDL> print,(*c[0.]).t
       0
IDL> (*c[0.]).t=9
IDL> print,c[0].t
       9
```

To make things like

c[0].t=1

valid, the IDL interpreter would have to change, so that when an
object with brackets shows up in the left side of the assignment but
it is qualified (with the .t, in this case) the object's
_overloadBracketsRightSide would be called, to return a variable, then
whatever that assignment does to the variable is performed, then at
the end the variable is passed back to the object's
_overloadBracketsLeftSide.

On Mar 15, 11:02 am, fawltylangu...@gmail.com wrote:
> On Friday, March 15, 2013 9:10:45 AM UTC+1, Tom Grydeland wrote:
>
>> IDL> c = List({ t: 0})
>> IDL> print, c[0].t
>>            0
>> IDL> c[0].t = 1

>> % Attempt to store into an expression: Structure reference.
>> % Execution halted at: $MAIN$
>
> This is a bug IMHO. This construct should work as a structure array works (LIST is a pointer array in disguise):
>
> IDL> a=replicate({t:0}, 1)
> IDL> a[0].t=1
> IDL> print, a[0].t
> 1
>
> regards,
> Lajos

---

Subject: Re: Storing !NULL in struct
Posted by tom.grydeland on Mon, 18 Mar 2013 09:00:58 GMT
View Forum Message <> Reply to Message

On Friday, March 15, 2013 8:40:56 PM UTC, Chris Torrence wrote:
> Hi all,

> Not to stir the pot some more [...]

Not at all, it is good to hear the rationale behind decisions such as this one.

> The deal-killer was that IDL structures are supposed to map directly to C structures. [...]

I'll buy that.

> As Mike suggests, perhaps HASH is the way to go.

Either of HASH or LIST would be perfectly fine, if I were able to even assign to already-known fields of structs stored inside them:

```
IDL> h = hash('f', {t:0})
IDL> help, h
H               HASH  <ID=198  NELEMENTS=1>
IDL> print, h['f'].t
          0
IDL> h['f'].t = 1
% Attempt to store into an expression: Structure reference.
% Execution halted at: $MAIN$
IDL> c = list({t:0})
IDL> print, c[0].t
          0
IDL> c[0].t = 1
% Attempt to store into an expression: Structure reference.
```

% Execution halted at: $MAIN$

It's not that I cannot imagine a way of working around this, but it seems to defeat the purpose of providing high-level data structures.

> Chris

--T

---

Subject: Re: Storing !NULL in struct
Posted by lecacheux.alain on Mon, 18 Mar 2013 09:27:31 GMT
View Forum Message <> Reply to Message

Le lundi 18 mars 2013 10:00:58 UTC+1, Tom Grydeland a écrit :
> On Friday, March 15, 2013 8:40:56 PM UTC, Chris Torrence wrote:
>
>> Hi all,
>
>
>
>> Not to stir the pot some more [...]
>
>
>
> Not at all, it is good to hear the rationale behind decisions such as this one.
>
>
>
>> The deal-killer was that IDL structures are supposed to map directly to C structures. [...]
>
>
>
> I'll buy that.
>
>
>
>> As Mike suggests, perhaps HASH is the way to go.
>
>
>
> Either of HASH or LIST would be perfectly fine, if I were able to even assign to already-known fields of structs stored inside them:
>
>
>
> IDL> h = hash('f', {t:0})

>
> IDL> help, h
>
> H              HASH  <ID=198  NELEMENTS=1>
>
> IDL> print, h['f'].t
>
>           0
>
> IDL> h['f'].t = 1
>
> % Attempt to store into an expression: Structure reference.
>
> % Execution halted at: $MAIN$
>
> IDL> c = list({t:0})
>
> IDL> print, c[0].t
>
>           0
>
> IDL> c[0].t = 1
>
> % Attempt to store into an expression: Structure reference.
>
> % Execution halted at: $MAIN$
>
>
>
>
>
> It's not that I cannot imagine a way of working around this, but it seems to defeat the purpose of providing high-level data structures.
>
>
>
>> Chris
>
>
>
> --T

IDL> h = hash('f', {t:0})
IDL> help, h
H              HASH  <ID=1  NELEMENTS=1>
IDL> print, h['f'].t
          0
IDL> h['f'] = {t:1}

IDL> print, h['f'].t
1

alx.

---

Subject: Re: Storing !NULL in struct
Posted by tom.grydeland on Mon, 18 Mar 2013 11:41:14 GMT
View Forum Message <> Reply to Message

On Monday, March 18, 2013 9:27:31 AM UTC, alx wrote:

> Le lundi 18 mars 2013 10:00:58 UTC+1, Tom Grydeland a écrit :
>> Either of HASH or LIST would be perfectly fine, if I were able to even assign to already-known fields of structs stored inside them:

>> It's not that I cannot imagine a way of working around this, but it seems to defeat the purpose of providing high-level data structures.

> IDL> h = hash('f', {t:0})
> IDL> help, h
> H HASH <ID=1 NELEMENTS=1>
> IDL> print, h['f'].t
> 0
> IDL> h['f'] = {t:1}
> IDL> print, h['f'].t
> 1

> alx.

Very good, so you, too, understand how to work _around_ this problem.

Do you also understand why I referred to this as defeating the purpose of high-level data structures?

--T

---

Subject: Re: Storing !NULL in struct
Posted by lecacheux.alain on Mon, 18 Mar 2013 12:20:51 GMT
View Forum Message <> Reply to Message

Le lundi 18 mars 2013 12:41:14 UTC+1, Tom Grydeland a écrit :
> On Monday, March 18, 2013 9:27:31 AM UTC, alx wrote:
>
>

>
>> Le lundi 18 mars 2013 10:00:58 UTC+1, Tom Grydeland a écrit :
>
>>> Either of HASH or LIST would be perfectly fine, if I were able to even assign to already-known fields of structs stored inside them:
>
>
>
>>> It's not that I cannot imagine a way of working around this, but it seems to defeat the purpose of providing high-level data structures.
>
>
>
>> IDL> h = hash('f', {t:0})
>
>> IDL> help, h
>
>> H               HASH  <ID=1  NELEMENTS=1>
>
>> IDL> print, h['f'].t
>
>>            0
>
>> IDL> h['f'] = {t:1}
>
>> IDL> print, h['f'].t
>
>>            1
>
>
>
>
>
>
>> alx.
>
>
>
> Very good, so you, too, understand how to work _around_ this problem.
>
>
>
> Do you also understand why I referred to this as defeating the purpose of high-level data structures?
>
>
>
> --T

> Do you also understand why I referred to this as defeating the purpose of high-level data structures?
No, I do'nt. I do not understand what you mean by "high-level" data structures.
As previously said, IDL structures are defined like C structures: each field being defined "by value". If you want more flexibility, you can (and you must) use pointers (each field is then defined "by reference").
What else ?
alx.

---

Subject: Re: Storing !NULL in struct
Posted by tom.grydeland on Mon, 18 Mar 2013 12:40:38 GMT
View Forum Message <> Reply to Message

On Monday, March 18, 2013 12:20:51 PM UTC, alx wrote:

>> Do you also understand why I referred to this as defeating the purpose of high-level data structures?

> No, I do'nt. I do not understand what you mean by "high-level" data structures.

Basically, LISTs and HASHes. These are capable of storing arbitrary values in each value slot, thereby giving you the opportunity to create arbitrarily complex nested structures. The problem is that values stored in these structures cannot easily be accessed and modified, which defeats the purpose of having these structures in the first place.

> As previously said, IDL structures are defined like C structures: each field being defined "by value". If you want more flexibility, you can (and you must) use pointers (each field is then defined "by reference").

This point has been made several times already, and in my reply to Chris Torrence, I granted that the reasons behind this decision is valid, even if it has a consequence that seems silly by itself.

What I was demonstrating in the message you replied to, however, is a different point. When a struct is stored in a LIST or a HASH, you cannot change its values, while a struct referred to by a pointer _can_ be modified.

Consider this simple example (and before you suggest workarounds, please consider that I already said I understand how to work around these issues, _AND_ that I'm deliberately constructing _simple_ examples, so that the points stand out more clearly, but that the cases I intend to deal with are going to be much more involved, and nested much more deeply than what I'm showing you here):

```
IDL> .run
- pro modify, s
-   s.foo += 1
- end
```

```
% Compiled module: MODIFY.
IDL> s = {foo:0}
IDL> modify, s
IDL> help, s
** Structure <23d4ed8>, 1 tags, length=4, data length=4, refs=1:
   FOO             LONG                 1

;; So a structure stored in a simple variable can be MODIFY'ed

IDL> p = ptr_new({foo:0})
IDL> modify, *p
IDL> help, *p
** Structure <23d4b08>, 1 tags, length=4, data length=4, refs=1:
   FOO             LONG                 1

;; Similarly, a structure stored behind a pointer can be MODIFY'ed

IDL> L = list({foo:0})
IDL> help, L[0]
** Structure <23ae368>, 1 tags, length=4, data length=4, refs=2:
   FOO             LONG                 0
IDL> modify, L[0]
IDL> help, L[0]
** Structure <23ae368>, 1 tags, length=4, data length=4, refs=2:
   FOO             LONG                 0

;; while a structure stored in a LIST _cannot_ be MODIFY'ed, and my example fails _without
warning_.
```

Do you see _now_ what I mean by defeating the purpose?

> alx.

--T

---

Subject: Re: Storing !NULL in struct
Posted by Michael Galloy on Mon, 18 Mar 2013 16:48:54 GMT
View Forum Message <> Reply to Message

On 3/18/13 6:40 AM, Tom Grydeland wrote:
> On Monday, March 18, 2013 12:20:51 PM UTC, alx wrote:
>
>>> Do you also understand why I referred to this as defeating the purpose of high-level data structures?
>
>> No, I do'nt. I do not understand what you mean by "high-level" data structures.
>

> Basically, LISTs and HASHes. These are capable of storing arbitrary values in each value slot, thereby giving you the opportunity to create arbitrarily complex nested structures. The problem is that values stored in these structures cannot easily be accessed and modified, which defeats the purpose of having these structures in the first place.
>
>> As previously said, IDL structures are defined like C structures: each field being defined "by value". If you want more flexibility, you can (and you must) use pointers (each field is then defined "by reference").
>
> This point has been made several times already, and in my reply to Chris Torrence, I granted that the reasons behind this decision is valid, even if it has a consequence that seems silly by itself.
>
> What I was demonstrating in the message you replied to, however, is a different point. When a struct is stored in a LIST or a HASH, you cannot change its values, while a struct referred to by a pointer _can_ be modified.
>
> Consider this simple example (and before you suggest workarounds, please consider that I already said I understand how to work around these issues, _AND_ that I'm deliberately constructing _simple_ examples, so that the points stand out more clearly, but that the cases I intend to deal with are going to be much more involved, and nested much more deeply than what I'm showing you here):
>
> IDL> .run
> - pro modify, s
> - s.foo += 1
> - end
> % Compiled module: MODIFY.
> IDL> s = {foo:0}
> IDL> modify, s
> IDL> help, s
> ** Structure <23d4ed8>, 1 tags, length=4, data length=4, refs=1:
> FOO LONG 1
>
> ;; So a structure stored in a simple variable can be MODIFY'ed
>
> IDL> p = ptr_new({foo:0})
> IDL> modify, *p
> IDL> help, *p
> ** Structure <23d4b08>, 1 tags, length=4, data length=4, refs=1:
> FOO LONG 1
>
> ;; Similarly, a structure stored behind a pointer can be MODIFY'ed
>
> IDL> L = list({foo:0})
> IDL> help, L[0]
> ** Structure <23ae368>, 1 tags, length=4, data length=4, refs=2:
> FOO LONG 0

> IDL> modify, L[0]
> IDL> help, L[0]
> ** Structure <23ae368>, 1 tags, length=4, data length=4, refs=2:
>  FOO  LONG  0
>
> ;; while a structure stored in a LIST _cannot_ be MODIFY'ed, and my example fails _without warning_.
>
> Do you see _now_ what I mean by defeating the purpose?
>
>> alx.
>
> --T
>

I think what you are showing here is that any variable passed by value
does not end up modified at the calling level. It doesn't work for
arrays of structures either:

```
IDL> sarr = replicate({ foo: 0 }, 10)
IDL> modify, sarr[0]
IDL> print, sarr[0]
{       0}
```

s and *p are named variables, but L[0] and sarr[0] are not. s and *p are
passed by reference, L[0] and sarr[0] have pass by value semantics.

My hash suggestion was to replace your structure with a hash:

```
IDL> c = list(hash('t', 0))
IDL> print, (c[0])['t']
       0
IDL> (c[0])['t'] = 1
IDL> print, (c[0])['t']
       1
```

Mike
--
Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)
Research Mathematician
Tech-X Corporation

---

Subject: Re: Storing !NULL in struct
Posted by Yngvar Larsen on Mon, 18 Mar 2013 18:26:51 GMT

On Monday, 18 March 2013 17:48:54 UTC+1, Mike Galloy wrote:
> On 3/18/13 6:40 AM, Tom Grydeland wrote:
>

>
> I think what you are showing here is that any variable passed by value
> does not end up modified at the calling level.

Right.

> It doesn't work for arrays of structures either:
>
> IDL> sarr = replicate({ foo: 0 }, 10)
> IDL> modify, sarr[0]
> IDL> print, sarr[0]
> { 0}

Yes, but you can still modify an element, or even a range of elements, of a structure array at your current calling level:

```
IDL> sarr[0].foo = 4
IDL> print, sarr[0].foo
      4
IDL> sarr[3:5].foo = 4
IDL> print, sarr[0:6].foo
      4       0       0       4       4       4       0
```

This is not the case for lists, as already discussed:

```
IDL> larr = list(length=10) & for n=0, 9 do larr[n] = {foo: 0}
IDL> print, larr[0].foo
      0
IDL> larr[0].foo = 4
% Attempt to store into an expression: Structure reference.
% Execution halted at: $MAIN$
```

I fail to see why the latter isn't allowed. This means that if you still want to modify structures within your list, you need to do something like this

```
IDL> tmp = larr[0]
IDL> tmp.foo = 4
IDL> larr[0] = tmp
IDL> print, larr[0].foo
      4
```

This seems silly to me. What is the purpose of not allowing this?

Switching to hash tables instead of structures as general purpose data structure is of course a very good  strategy in 2013, but I have at least 10 years worth of legacy code which already heavily (mis-)uses the anonymous structure as a "hash table light" with case insensitive keys.

--
Yngvar

---

Subject: Re: Storing !NULL in struct
Posted by Michael Galloy on Mon, 18 Mar 2013 19:33:47 GMT
View Forum Message <> Reply to Message

On 3/18/13 12:26 PM, Yngvar Larsen wrote:
>
> This is not the case for lists, as already discussed:
>
> IDL> larr = list(length=10) & for n=0, 9 do larr[n] = {foo: 0}
> IDL> print, larr[0].foo
>  0
> IDL> larr[0].foo = 4
> % Attempt to store into an expression: Structure reference.
> % Execution halted at: $MAIN$
>
> I fail to see why the latter isn't allowed.

I agree.

> This means that if you still want to modify structures within your list, you need to do something like this
>
> IDL> tmp = larr[0]
> IDL> tmp.foo = 4
> IDL> larr[0] = tmp
> IDL> print, larr[0].foo
>  4
>
> This seems silly to me. What is the purpose of not allowing this?
>
> Switching to hash tables instead of structures as general purpose data structure is of course a very good  strategy in 2013, but I have at least 10 years worth of legacy code which already heavily (mis-)uses the anonymous structure as a "hash table light" with case insensitive keys.

I agree here as well. But, I do find it difficult to use new IDL
features for long after they are no longer new since so many users do
not have the latest version of IDL.

Mike
--

Michael Galloy
www.michaelgalloy.com
Modern IDL: A Guide to IDL Programming (http://modernidl.idldev.com)
Research Mathematician
Tech-X Corporation

Subject: Re: Storing !NULL in struct
Posted by Bob[4] on Tue, 19 Mar 2013 03:26:50 GMT
View Forum Message <> Reply to Message

On Monday, March 18, 2013 12:26:51 PM UTC-6, Yngvar Larsen wrote:
> On Monday, 18 March 2013 17:48:54 UTC+1, Mike Galloy wrote:
>
>> On 3/18/13 6:40 AM, Tom Grydeland wrote:
>
>>
>
>
>
>>
>
>> I think what you are showing here is that any variable passed by value
>
>> does not end up modified at the calling level.
>
>
>
> Right.
>
>
>
>> It doesn't work for arrays of structures either:
>
>>
>
>> IDL> sarr = replicate({ foo: 0 }, 10)
>
>> IDL> modify, sarr[0]
>
>> IDL> print, sarr[0]
>
>> { 0}
>
>
>
> Yes, but you can still modify an element, or even a range of elements, of a structure array at your current calling level:

>
>
>
> IDL> sarr[0].foo = 4
>
> IDL> print, sarr[0].foo
>
>       4
>
> IDL> sarr[3:5].foo = 4
>
> IDL> print, sarr[0:6].foo
>
>       4      0      0      4      4      4      0
>
>
>
> This is not the case for lists, as already discussed:
>
>
>
> IDL> larr = list(length=10) & for n=0, 9 do larr[n] = {foo: 0}
>
> IDL> print, larr[0].foo
>
>       0
>
> IDL> larr[0].foo = 4
>
> % Attempt to store into an expression: Structure reference.
>
> % Execution halted at: $MAIN$
>
>
>
> I fail to see why the latter isn't allowed.

I agree. LISTs and HASHes should return a reference to their elements and not a temporary copy.

In addition, IDL needs a reference type that would be similar to a PTR but would not need to be de-referenced to get at what it is pointing at. It could use de-referencing (or a function call) to set the reference but then would allow syntax like a normal variable. This would greatly simplify IDL programing and could perhaps be used to fix the mess that IDL LISTs and HASHes are.

---

Subject: Re: Storing !NULL in struct
Posted by Yngvar Larsen on Tue, 19 Mar 2013 08:59:39 GMT

On Tuesday, 19 March 2013 04:26:50 UTC+1, bobnn...@gmail.com wrote:

> I agree. LISTs and HASHes should return a reference to their elements and not a temporary copy.

Well put. This single sentence summarizes (my position in) the discussion well.

> In addition, IDL needs a reference type that would be similar to a PTR but would not need to be de-referenced to get at what it is pointing at. It could use de-referencing (or a function call) to set the reference but then would allow syntax like a normal variable. This would greatly simplify IDL programing and could perhaps be used to fix the mess that IDL LISTs and HASHes are.

I often use C =TEMPORARY(*P) in these situations, but then the contents of the pointer is then left undefined, and needs to be put back after use with P = PTR_NEW(C, /NO_COPY). Your suggestion would make this much easier.

--
Yngvar