Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by chris_torrence@NOSPAM on Mon, 21 Oct 2013 23:07:41 GMT
View Forum Message <> Reply to Message

Hi Paul,

Nothing has changed with the way IDL passes objects. However, I'm a little confused by your code. When you say that the "compute_interpolation_frequency" procedure "allocates the resulting object", do you really mean that it just fills in some properties on that object? Because it looks like you are doing an obj_new on those objects before passing them in.

It looks like something strange is going on with garbage collection, where it is somehow freeing up your object inside compute_interpolation_frequency. However, I can't imagine why this would be happening. I just create a test program which approximates what you are doing:

```
pro test_pass_objelement, obj
  obj->getproperty, name = name
  obj->SetProperty, NAME='NewName'
end
o = objarr(5)
for i=0,4 do begin
  o[i] = obj_new('IDLitComponent', NAME=STRTRIM(i,2))
  test_pass_objelement, o[i]
  print, obj_valid(o[i])
endfor
end
```

When I run this code (at least in IDL 8.3), the objects are all valid after the procedure call. Can you try running this code to make sure it passes for you? If it does, then maybe you can post the details of your compute_interpolation_frequency procedure, so we can diagnose what is happening inside

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by Paul Van Delst[1] on Tue, 22 Oct 2013 12:16:30 GMT
View Forum Message <> Reply to Message

Hi Chris,

Your test code works fine for me. But I changed it to more closely
approximate what I am doing (and added more output of the properties).

The important change is that the procedure in question has been changed
to a method.

```
-----%<-----
pro IDLitComponent::test_pass_objelement, obj
   obj = obj_new('IDLitComponent')
```

```
   self->GetProperty, name=name
   obj->SetProperty, NAME=name       ; Set the name...
   obj->GetProperty, NAME=name       ; ...get it...
   print, "In method. Name = ", name  ; ...and print
end

pro test_pass
o = objarr(5)
for i=0,4 do begin
   scalarobj = obj_new('IDLitComponent',name='scalar work obj')
   o[i]      = obj_new('IDLitComponent',name='empty rank-1')
   scalarobj->test_pass_objelement, o[i]
   o[i]->getproperty, name = name     ; Get the name...
   print, "In caller. Name = ", name  ; ...and print
   print
endfor
end
-----%<-----
```

When I run "test_pass" I get the following output:

```
IDL> test_pass
% Compiled module: TEST_PASS.
In method. Name = scalar work obj
In caller. Name = empty rank-1

In method. Name = scalar work obj
In caller. Name = empty rank-1

In method. Name = scalar work obj
In caller. Name = empty rank-1

In method. Name = scalar work obj
In caller. Name = empty rank-1

In method. Name = scalar work obj
In caller. Name = empty rank-1
```

So, while the *passed* object is valid upon return, all of the changes made to it in the method didn't "take".

This is only a problem when the object is an argument to one of its own methods, but invoked via a different object (in this case "scalarobj").

When the object is passed as an argument to a "regular" procedure (as in your test case) everything works as expected.

Any ideas? The actual code in question (my operational equivalent of the main "test_pass") has existed since 2011 - and it's been used to process data for several satellite sensors.

cheers,

paulv

On 10/21/2013 07:07 PM, Chris Torrence wrote:
> Hi Paul,
>
> Nothing has changed with the way IDL passes objects. However, I'm a
> little confused by your code. When you say that the
> "compute_interpolation_frequency" procedure "allocates the resulting
> object", do you really mean that it just fills in some properties on
> that object? Because it looks like you are doing an obj_new on those
> objects before passing them in.
>
> It looks like something strange is going on with garbage collection,
> where it is somehow freeing up your object inside
> compute_interpolation_frequency. However, I can't imagine why this
> would be happening. I just create a test program which approximates
> what you are doing:
>
> pro test_pass_objelement, obj obj->getproperty, name = name
> obj->SetProperty, NAME='NewName' end o = objarr(5) for i=0,4 do
> begin o[i] = obj_new('IDLitComponent', NAME=STRTRIM(i,2))
> test_pass_objelement, o[i] print, obj_valid(o[i]) endfor end
>
> When I run this code (at least in IDL 8.3), the objects are all valid
> after the procedure call. Can you try running this code to make sure
> it passes for you? If it does, then maybe you can post the details of
> your compute_interpolation_frequency procedure, so we can diagnose
> what is happening inside.
>
> Thanks! -Chris ExelisVIS
>

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by Paul Van Delst[1] on Tue, 22 Oct 2013 13:03:49 GMT
View Forum Message <> Reply to Message

O.k. I think I've figured out a workaround -- but I would still like to know why the test case I posted doesn't work.

I went through our repository logs and saw that I recently added the

creation of a new object in the method.

So, if I remove the line
   obj = obj_new('IDLitComponent')
from the IDLitComponent::test_pass_objelement method, everything works
as expected:

IDL> test_pass
% Compiled module: TEST_PASS.
In method. Name = scalar work obj
In caller. Name = scalar work obj

In method. Name = scalar work obj
In caller. Name = scalar work obj

In method. Name = scalar work obj
In caller. Name = scalar work obj

In method. Name = scalar work obj
In caller. Name = scalar work obj

In method. Name = scalar work obj
In caller. Name = scalar work obj

So, are we dealing with pass-by-reference/value confusion on my part here?

That is, the creation of new object in the method doesn't do anything
about the reference to the original object that was passed in, and
eventually returned back to the caller?

cheers,

paulv

On 10/22/2013 08:16 AM, Paul van Delst wrote:
> Hi Chris,
>
> Your test code works fine for me. But I changed it to more closely
> approximate what I am doing (and added more output of the properties).
>
> The important change is that the procedure in question has been changed
> to a method.
>
> -----%<-----
> pro IDLitComponent::test_pass_objelement, obj
>   obj = obj_new('IDLitComponent')

```
>    self->GetProperty, name=name
>    obj->SetProperty, NAME=name       ; Set the name...
>    obj->GetProperty, NAME=name       ; ...get it...
>    print, "In method. Name = ", name  ; ...and print
> end
>
> pro test_pass
> o = objarr(5)
> for i=0,4 do begin
>    scalarobj = obj_new('IDLitComponent',name='scalar work obj')
>    o[i]      = obj_new('IDLitComponent',name='empty rank-1')
>    scalarobj->test_pass_objelement, o[i]
>    o[i]->getproperty, name = name     ; Get the name...
>    print, "In caller. Name = ", name  ; ...and print
>    print
> endfor
> end
> -----%<-----
>
> When I run "test_pass" I get the following output:
>
> IDL> test_pass
> % Compiled module: TEST_PASS.
> In method. Name = scalar work obj
> In caller. Name = empty rank-1
>
> In method. Name = scalar work obj
> In caller. Name = empty rank-1
>
> In method. Name = scalar work obj
> In caller. Name = empty rank-1
>
> In method. Name = scalar work obj
> In caller. Name = empty rank-1
>
> In method. Name = scalar work obj
> In caller. Name = empty rank-1
>
>
> So, while the *passed* object is valid upon return, all of the changes
> made to it in the method didn't "take".
>
> This is only a problem when the object is an argument to one of its own
> methods, but invoked via a different object (in this case "scalarobj").
>
> When the object is passed as an argument to a "regular" procedure (as in
> your test case) everything works as expected.
>
```

> Any ideas? The actual code in question (my operational equivalent of the
> main "test_pass") has existed since 2011 - and it's been used to process
> data for several satellite sensors.
>
> cheers,
>
> paulv
>
>
> On 10/21/2013 07:07 PM, Chris Torrence wrote:
>> Hi Paul,
>>
>> Nothing has changed with the way IDL passes objects. However, I'm a
>> little confused by your code. When you say that the
>> "compute_interpolation_frequency" procedure "allocates the resulting
>> object", do you really mean that it just fills in some properties on
>> that object? Because it looks like you are doing an obj_new on those
>> objects before passing them in.
>>
>> It looks like something strange is going on with garbage collection,
>> where it is somehow freeing up your object inside
>> compute_interpolation_frequency. However, I can't imagine why this
>> would be happening. I just create a test program which approximates
>> what you are doing:
>>
>> pro test_pass_objelement, obj obj->getproperty, name = name
>> obj->SetProperty, NAME='NewName' end o = objarr(5) for i=0,4 do
>> begin o[i] = obj_new('IDLitComponent', NAME=STRTRIM(i,2))
>> test_pass_objelement, o[i] print, obj_valid(o[i]) endfor end
>>
>> When I run this code (at least in IDL 8.3), the objects are all valid
>> after the procedure call. Can you try running this code to make sure
>> it passes for you? If it does, then maybe you can post the details of
>> your compute_interpolation_frequency procedure, so we can diagnose
>> what is happening inside.
>>
>> Thanks! -Chris ExelisVIS
>>

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by David Fanning on Tue, 22 Oct 2013 13:39:00 GMT
View Forum Message <> Reply to Message

Paul van Delst writes:

> That is, the creation of new object in the method doesn't do anything
> about the reference to the original object that was passed in, and

> eventually returned back to the caller?

This appears to be the semi-passed-by-reference condition. ;-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by Paul Van Delst[1] on Tue, 22 Oct 2013 14:10:44 GMT
View Forum Message <> Reply to Message

Ha!

It's probably my advancing age, but these special cases with regards to argument passing are really starting to annoy me... mostly because I *still* keep brain-fading about them, but also because these days I expect my code to do what I want, rather than what I tell it to do - a definite sign of old-codger-ism[*] :o).

Time to translate the code to Fortran2003 I guess. Urg. (Which we sorta have to do anyway, I was just hoping not to need to do it for a couple more years.)

cheers,

paulv

[*] Or, more likely, I think I'm in an "autonomous stage" but I'm really still in a "cognitive stage" with regards to programming skill.
(http://www.brainpickings.org/index.php/2013/10/17/ok-plateau)

On 10/22/2013 09:39 AM, David Fanning wrote:
> Paul van Delst writes:
>
>> That is, the creation of new object in the method doesn't do anything
>> about the reference to the original object that was passed in, and
>> eventually returned back to the caller?
>

> This appears to be the semi-passed-by-reference condition. ;-)
>
> Cheers,
>
> David
>
>
>

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by David Fanning on Tue, 22 Oct 2013 14:43:29 GMT
View Forum Message <> Reply to Message

Paul van Delst writes:

> It's probably my advancing age

Well, if you are like me, there is no doubt about this. :-)

This situation sorta makes sense to me, though.

I think subscripted arrays are passed by value, not by reference. But, in the case of an object, the *fields* of the object are often pointers and other objects (i.e., heap variables) that act like normal variables and are passed by reference. So, you really do find yourself in this sort of quantum state where the cat is both alive and dead at the same time. :-)

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by David Fanning on Tue, 22 Oct 2013 14:51:23 GMT
View Forum Message <> Reply to Message

Paul van Delst writes:

> [*] Or, more likely, I think I'm in an "autonomous stage" but I'm really
> still in a "cognitive stage" with regards to programming skill.
> (http://www.brainpickings.org/index.php/2013/10/17/ok-plateau)

Humm. Probably explains why my backhand is so damn ineffective. :-(

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by chris_torrence@NOSPAM on Tue, 22 Oct 2013 15:57:16 GMT
View Forum Message <> Reply to Message

On Tuesday, October 22, 2013 8:10:44 AM UTC-6, Paul van Delst wrote:
> Ha!
>
>
>
> It's probably my advancing age, but these special cases with regards to
>
> argument passing are really starting to annoy me... mostly because I
>
> *still* keep brain-fading about them, but also because these days I
>
> expect my code to do what I want, rather than what I tell it to do - a
>
> definite sign of old-codger-ism[*] :o).
>
>
>
> Time to translate the code to Fortran2003 I guess. Urg. (Which we sorta
>
> have to do anyway, I was just hoping not to need to do it for a couple
>
> more years.)
>
>

>
> cheers,
>
>
>
> paulv
>
>
>
> [*] Or, more likely, I think I'm in an "autonomous stage" but I'm really
>
> still in a "cognitive stage" with regards to programming skill.
>
> (http://www.brainpickings.org/index.php/2013/10/17/ok-plateau)
>
>
>
> On 10/22/2013 09:39 AM, David Fanning wrote:
>
>> Paul van Delst writes:
>
>>
>
>>> That is, the creation of new object in the method doesn't do anything
>
>>> about the reference to the original object that was passed in, and
>
>>> eventually returned back to the caller?
>
>>
>
>> This appears to be the semi-passed-by-reference condition. ;-)
>
>>
>
>> Cheers,
>
>>
>
>> David
>
>>
>
>>
>
>>

Hi Paul,

I don't think this is a special case. It's also not "semi-passed-by-reference." The code is simply passing an expression into a routine, so IDL cannot store into it.
Cheers,
Chris

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by David Fanning on Tue, 22 Oct 2013 16:47:57 GMT
View Forum Message <> Reply to Message

Chris Torrence writes:

> I don't think this is a special case. It's also not "semi-passed-by-reference." The code is simply passing an expression into a routine, so IDL cannot store into it.

The point is, it *does* store into it, even though it is being passed by value!

The object reference is passed by value, but the object itself, as a heap variable, is more like a variable that is passed by reference. This gives this situation the semi-passed-by-reference feel.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thue. ("Perhaps thou speakest truth.")

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by Paul Van Delst[1] on Tue, 22 Oct 2013 16:59:18 GMT
View Forum Message <> Reply to Message

On 10/22/2013 11:57 AM, Chris Torrence wrote:
>
> Hi Paul,
> I don't think this is a special case. It's also not
> "semi-passed-by-reference." The code is simply passing an expression
> into a routine, so IDL cannot store into it.
> Cheers, Chris

Well, that's not the case. Both your, and my corrected, test cases shows that you *can* store into the array element reference (well, the object that was referenced in that array element) in the callee.

It was the "redefinition" of the object in the routine (or method) that
was screwing things up.

I know it won't (can't) change anytime soon, but I find the fact that an
array element reference is considered an expression in IDL very
confusing. It just doesn't grok well.

IDL users should be shielded from under-the-hood details like argument
passing mechanisms, IMO (that's my Fortran90/95/2003 side talking).

cheers,

paulv

Subject: Re: object argument passing behaviour changed in v8.2.2?
Posted by chris_torrence@NOSPAM on Tue, 22 Oct 2013 17:17:00 GMT
View Forum Message <> Reply to Message

On Tuesday, October 22, 2013 10:59:18 AM UTC-6, Paul van Delst wrote:
> On 10/22/2013 11:57 AM, Chris Torrence wrote:
>
>>
>
>> Hi Paul,
>
>> I don't think this is a special case. It's also not
>
>> "semi-passed-by-reference." The code is simply passing an expression
>
>> into a routine, so IDL cannot store into it.
>
>> Cheers, Chris
>
>
>
> Well, that's not the case. Both your, and my corrected, test cases shows
>
> that you *can* store into the array element reference (well, the object
>
> that was referenced in that array element) in the callee.
>
>
>
> It was the "redefinition" of the object in the routine (or method) that
>
> was screwing things up.

> I know it won't (can't) change anytime soon, but I find the fact that an
> array element reference is considered an expression in IDL very
> confusing. It just doesn't grok well.
> IDL users should be shielded from under-the-hood details like argument
> passing mechanisms, IMO (that's my Fortran90/95/2003 side talking).
> cheers,
> paulv

Yep, I agree that a different decision would have been better. Unfortunately, we would have to travel back 30 years in time to tell David Stern... If I recall, Fortran 77 passed by reference (even for array elements), so David could have gotten it right... Oh well.
-Chris
