Subject: Re: _extra and call_method
Posted by David Fanning on Fri, 28 Jan 2005 01:00:06 GMT
View Forum Message <> Reply to Message

Benjamin Hornberger writes:

> can't keyword passing by _extra be used with call_method? I am getting
> an error message "Keyword parameters not allowed in call" if I try to do
> so.
>
> I want to write a wrapper routine which first checks if an object is
> valid, and if yes, calls methods on it. For positional parameters, I
> have to use "case n_params() of", which is not very nice, but for
> keywords at least I would like to use inheritance...

Are you *sure* you are doing this correctly? I've never
had the least bit of trouble with it. It sounds to me
like the method you are calling does not have the particular
keyword you are using defined for it. Is this a spelling
problem, perhaps?

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: _extra and call_method
Posted by Benjamin Hornberger on Fri, 28 Jan 2005 20:38:11 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Benjamin Hornberger writes:
>
>
>> can't keyword passing by _extra be used with call_method? I am getting
>> an error message "Keyword parameters not allowed in call" if I try to do
>> so.
>>
>> I want to write a wrapper routine which first checks if an object is
>> valid, and if yes, calls methods on it. For positional parameters, I
>> have to use "case n_params() of", which is not very nice, but for
>> keywords at least I would like to use inheritance...
>

>
> Are you *sure* you are doing this correctly? I've never
> had the least bit of trouble with it. It sounds to me
> like the method you are calling does not have the particular
> keyword you are using defined for it. Is this a spelling
> problem, perhaps?
>
> Cheers,
>
> David
>

Well, I figured out what the problem is. It's not just about
call_method, but keyword inheritance in general. If you call a procedure
and pass _extra keywords to it even though it does not accept any
keywords at all, the error occurs. Simple example:

```
PRO taco

  print, 'Hello, world!'

END

PRO call_taco, _extra=extra

  taco, _extra=extra

END
```

In this case, when running "call_taco", IDL issues an error because I
call "taco" with _extra keywords even though it doesn't accept any.

In my case, I want to write a general wrapper routine for several
methods of an object. For simplicity, let's say all object methods
accept one positional parameter, but zero or more keywords. What I would
like to do is:

```
PRO call_object, object, method, par, _extra=extra

  ;; method is passed as a string

  IF ~obj_valid(object) THEN BEGIN
    ;; issue an error and more
  ENDIF

  call_method, method, object, par, _extra=extra

END
```

Now if there are any object methods which don't have any keywords
defined, the error occurs as described.

Any more hints, except defining a dummy keyword for each method?

Thanks,
Benjamin

Subject: Re: _extra and call_method
Posted by David Fanning on Fri, 28 Jan 2005 22:21:00 GMT

View Forum Message <> Reply to Message

Benjamin Hornberger writes:

> Any more hints, except defining a dummy keyword for each method?

I pretty much make _Extra a feature of every procedure or
function definition statement I write. It never hurts, and
sometimes helps. :-)

Similarly, I've gotten in the habit of returning
the dummy structure as an output argument for every
object I create. That *really* comes in handy!

```
 PRO myobject__define, class
  class = {MYCLASS, myvar:0}
 END
```

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: _extra and call_method
Posted by btt on Mon, 31 Jan 2005 13:13:39 GMT

View Forum Message <> Reply to Message

David Fanning wrote:
> Benjamin Hornberger writes:
>
>

>> Any more hints, except defining a dummy keyword for each method?
>
>
> I pretty much make _Extra a feature of every procedure or
> function definition statement I write. It never hurts, and
> sometimes helps. :-)
>
> Similarly, I've gotten in the habit of returning
> the dummy structure as an output argument for every
> object I create. That *really* comes in handy!
>

```
>   PRO myobject__define, class
>    class = {MYCLASS, myvar:0}
>   END
>
```

Hi,

I'm running on just one cup of coffee this morning so maybe this is a fuzzy
question: could you explain the circumstances in which this is useful? If you
return dummy named structure - well, what about all the work that goes into
populating its properties it via the INIT function? Or is this for simple data
structures (ala widget event structures, etc. ?)

Thanks,
Ben

Subject: Re: _extra and call_method
Posted by David Fanning on Mon, 31 Jan 2005 14:41:50 GMT
View Forum Message <> Reply to Message

Ben Tupper writes:

> I'm running on just one cup of coffee this morning so maybe this is a fuzzy
> question: could you explain the circumstances in which this is useful? If you
> return dummy named structure - well, what about all the work that goes into
> populating its properties it via the INIT function? Or is this for simple data
> structures (ala widget event structures, etc. ?)

Since objects are implemented as named structures in IDL,
I seem to find a number of instances where it would be
helpful to know what the names of the fields in that object
structure are. For example, one of the hugely time-consuming
tasks in object writing is creating the GetProperty and SetProperty
methods that allow you to manipulate and set/get values in the
object structure. Wouldn't it be nice to automate those tasks
and be able to get and set any property (field) in the object

without necessarily knowing ahead of time what those properties
might be? For example, I might like to respond to this:

```
   anObject -> SetProperty, Foo=5
```

Without specifically having to define the FOO keyword for the object.

If FOO were a field of this object, I could write a generic SetProperty
method like this (I'm leaving out a couple of important details, but I
plan an article soon):

```
PRO myObject::SetProperty, _Extra=extra

   ; What keywords are you looking for?
   keywords = Tag_Names(_extra)

   ; What properties (fields) can be changed?
   Call_Procedure, Obj_Class(self) + '__define', struct ;**************
   properties = Tag_Names(struct)

   ; Set the value of each field according to the keyword value.
   FOR j=0,N_Elements(keywords)-1 DO
      propertyIndex = Where(StrPos(properties, keywords[j]) EQ 0, match)
      IF match EQ 1 THEN self.(propertyIndex) = _extra.(j)
   ENDFOR

END
```

I can do something similar for a GetProperty method. Adding (copying,
really) these two generic methods to every object I create, is MUCH
less time consuming than defining each and every keyword for each
and every property I hope to change.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: _extra and call_method
Posted by btt on Mon, 31 Jan 2005 15:43:24 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

```
>  Ben Tupper writes:
>
>
>> I'm running on just one cup of coffee this morning so maybe this is a fuzzy
>> question: could you explain the circumstances in which this is useful?  If you
>> return dummy named structure - well, what about all the work that goes into
>> populating its properties it via the INIT function?  Or is this for simple data
>> structures (ala widget event structures, etc. ?)
>
>
>  Since objects are implemented as named structures in IDL,
>  I seem to find a number of instances where it would be
>  helpful to know what the names of the fields in that object
>  structure are. For example, one of the hugely time-consuming
>  tasks in object writing is creating the GetProperty and SetProperty
>  methods that allow you to manipulate and set/get values in the
>  object structure. Wouldn't it be nice to automate those tasks
>  and be able to get and set any property (field) in the object
>  without necessarily knowing ahead of time what those properties
>  might be? For example, I might like to respond to this:
>
>    anObject -> SetProperty, Foo=5
>
>  Without specifically having to define the FOO keyword for the object.
>
>  If FOO were a field of this object, I could write a generic SetProperty
>  method like this (I'm leaving out a couple of important details, but I
>  plan an article soon):
>
>  PRO myObject::SetProperty, _Extra=extra
>
>    ; What keywords are you looking for?
>    keywords = Tag_Names(_extra)
>
>    ; What properties (fields) can be changed?
>    Call_Procedure, Obj_Class(self) + '__define', struct ;*************
>    properties = Tag_Names(struct)
>
>    ; Set the value of each field according to the keyword value.
>    FOR j=0,N_Elements(keywords)-1 DO
>       propertyIndex = Where(StrPos(properties, keywords[j]) EQ 0, match)
>       IF match EQ 1 THEN self.(propertyIndex) = _extra.(j)
>    ENDFOR
>
>  END
>
>  I can do something similar for a GetProperty method. Adding (copying,
```

> really) these two generic methods to every object I create, is MUCH
> less time consuming than defining each and every keyword for each
> and every property I hope to change.
>
> Cheers,
>
> David
>

Oh neat! Thanks, David. I recall that you have mentioned this before. Ben

---

Subject: Re: _extra and call_method
Posted by btt on Tue, 01 Feb 2005 15:18:25 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
> Ben Tupper writes:
>
>
>> I'm running on just one cup of coffee this morning so maybe this is a fuzzy
>> question: could you explain the circumstances in which this is useful? If you
>> return dummy named structure - well, what about all the work that goes into
>> populating its properties it via the INIT function? Or is this for simple data
>> structures (ala widget event structures, etc. ?)
>
>
> Since objects are implemented as named structures in IDL,
> I seem to find a number of instances where it would be
> helpful to know what the names of the fields in that object
> structure are. For example, one of the hugely time-consuming
> tasks in object writing is creating the GetProperty and SetProperty
> methods that allow you to manipulate and set/get values in the
> object structure. Wouldn't it be nice to automate those tasks
> and be able to get and set any property (field) in the object
> without necessarily knowing ahead of time what those properties
> might be? For example, I might like to respond to this:
>
>   anObject -> SetProperty, Foo=5
>
> Without specifically having to define the FOO keyword for the object.
>
> If FOO were a field of this object, I could write a generic SetProperty
> method like this (I'm leaving out a couple of important details, but I
> plan an article soon):
>
> PRO myObject::SetProperty, _Extra=extra
>

```
>     ; What keywords are you looking for?
>     keywords = Tag_Names(_extra)
>
>     ; What properties (fields) can be changed?
>     Call_Procedure, Obj_Class(self) + '__define', struct ;*************
>     properties = Tag_Names(struct)
>
>     ; Set the value of each field according to the keyword value.
>     FOR j=0,N_Elements(keywords)-1 DO
>        propertyIndex = Where(StrPos(properties, keywords[j]) EQ 0, match)
>        IF match EQ 1 THEN self.(propertyIndex) = _extra.(j)
>     ENDFOR
>
>  END
>
>  I can do something similar for a GetProperty method. Adding (copying,
>  really) these two generic methods to every object I create, is MUCH
>  less time consuming than defining each and every keyword for each
>  and every property I hope to change.
>
```

Hi again,

Just got to thinking on this some more. Here's one item I thought you might be willing to share your thoughts on: in the case of multiple inheritances does each keyword get checked in for each generation of inheritance and if they do, does it matter?

For example...

```
PRO APPENDAGE__DEFINE, class
class = {APPENDAGE, isJointed:0}
END

PRO LEG__DEFINE, class
class = {ARM, INHERITS APPENDAGE, hasOpposingThumb: 0}
END
```

So, if I use the generic keyword checking for LEG does "isJointed" get checked twice? Propbably it doesn't matter a hoot if they do get checked twice.

```
PRO LEG::SetProperty, _EXTRA = extra

... do that neat keyword checking thing here ...
```

self->APPENDAGE::SetProperty, _Extra = extra
END

Cheers,
Ben

Subject: Re: _extra and call_method
Posted by David Fanning on Tue, 01 Feb 2005 15:40:16 GMT
View Forum Message <> Reply to Message

Ben Tupper writes:

```
>  Just got to thinking on this some more.  Here's one item I thought you might be
>  willing to share your thoughts on: in the case of multiple inheritances does
>  each keyword get checked in for each generation of inheritance and if they do,
>  does it matter?
>
>  For example...
>
>  PRO APPENDAGE__DEFINE, class
>  class = {APPENDAGE, isJointed:0}
>  END
>
>  PRO LEG__DEFINE, class
>  class = {ARM, INHERITS APPENDAGE, hasOpposingThumb: 0}
>  END
>
>
>  So, if I use the generic keyword checking for LEG does "isJointed" get checked
>  twice?  Propbably it doesn't matter a hoot if they do get checked twice.
>
>
>
>  PRO LEG::SetProperty, _EXTRA = extra
>
>  ... do that neat keyword checking thing here ...
>
>  self->APPENDAGE::SetProperty, _Extra = extra
>  END
```

The generic methods short-circuit the normal chaining of
keywords through SetProperty methods (could be a bad thing,
I guess, in the wrong hands) by setting the structure field
of self directly. If the field is there, it gets set. End of
story.

Of course, not all properties act this way. Sometimes when you
set one property, you are obliged to set others, etc. This
mechanism doesn't help with that, and, in fact, probably shouldn't
be used if you are doing things like that.

I use the generic methods mostly for quick and dirty objects,
or for when I am in the "development as thinking" stage of a
project. For the real-deal objects, I prefer to define the keywords,
since then I have the chance to document them. Makes it easier for
the end-user. :-)

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/