
Subject: how to get info on object fields

Posted by [Mirko Vukovic](#) on Mon, 25 Aug 1997 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hi,

I was trying to write a general extraction routine for objects. The idea was to pass the name of a field, it would check if the field exists, get its tag number and extract it. It would work on the object as if it were a structure. But TAG_NAMES does not work on objects (which does make sense as SELF is still an object (type =11), not a structure). (see code below for my implementation).

I like using objects, but sometimes they are too cumbersome for debugging purposes. I would also like to have a set of general routines applicable to all objects (via inheritance). That is why a general routine for extracting a field would be nice. Any clues how I might go about it?

```
FUNCTION structured::extract,arg,depointer = depointer
```

```
; check if arg is a valid field name
```

```
  ii = where(tag_names(self) EQ strupcase(arg),count)
```

```
  IF count NE 0 THEN begin
```

```
    type = size(self.(ii))
```

```
    type = type[type[0]+1]
```

```
    IF keyword_set(depointer) THEN if type EQ 10 THEN
```

```
return,*self.(ii)
```

```
  return,self.(ii)
```

```
ENDIF ELSE BEGIN
```

```
  message,/info,'Invalid field name provided'
```

```
  print,'You specified'+arg
```

```
  print,'Available fields are:'
```

```
  print,tag_names(self)
```

```
  return,0
```

```
ENDELSE
```

--

Mirko Vukovic, Ph.D 3075 Hansen Way M/S K-109

Novellus Systems Palo Alto, CA, 94304

415/424-4969 mirko.vukovic@varian.grc.com

Subject: Re: how to get info on object fields

Posted by [davidf](#) on Mon, 25 Aug 1997 07:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

Mirko Vukovic writes:

> I was trying to write a general extraction routine for objects. The
> idea was to pass the name of a field, it would check if the field
> exists, get its tag number and extract it. It would work on the object
> as if it were a structure. But TAG_NAMES does not work on objects
> (which does make sense as SELF is still an object (type =11), not a
> structure). (see code below for my implementation).
>
> I like using objects, but sometimes they are too cumbersome for
> debugging purposes. I would also like to have a set of general routines
> applicable to all objects (via inheritance). That is why a general
> routine for extracting a field would be nice. Any clues how I might go
> about it?

I used this code to get something working in just a few minutes. These objects can have superclasses, of course, and I am not checking for that. You could use Obj_Class to find all the superclass structures in an iterative way. The code would not be too hard to implement.

The way this code works is to use Obj_Class to get the name of the object, which is the name of the structure you want. I create the right kind of structure with the Execute command. To extract the field, I have to use a second Execute statement.

```
FUNCTION OBJECT::EXTRACT, field
```

```
; Check if FIELD is a valid field name. If it is,  
; return the field. If not, return -1.
```

```
thisClass = Obj_Class(self)  
ok = Execute('thisStruct = {' + thisClass + '}')  
structFields = Tag_Names(thisStruct)  
index = WHERE(structFields EQ StrUpCase(field), count)  
IF count EQ 1 THEN BEGIN  
  ok = Execute('Return, SELF.' + structFields(index(0)))  
ENDIF ELSE BEGIN  
  Message, 'Can not find field "' + field + $  
  ' in structure.', /Informational  
  RETURN, -1  
ENDELSE  
END
```

This should give you some ideas.

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
Customizable IDL Programming Courses
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com

Subject: Re: how to get info on object fields
Posted by [J.D. Smith](#) on Thu, 28 Aug 1997 07:00:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

David Fanning wrote:

>
> Mirko Vukovic writes:
>
>> I was trying to write a general extraction routine for objects. The
>> idea was to pass the name of a field, it would check if the field
>> exists, get its tag number and extract it. It would work on the object
>> as if it were a structure. But TAG_NAMES does not work on objects
>> (which does make sense as SELF is still an object (type =11), not a
>> structure). (see code below for my implementation).
>>
>> I like using objects, but sometimes they are too cumbersome for
>> debugging purposes. I would also like to have a set of general routines
>> applicable to all objects (via inheritance). That is why a general
>> routine for extracting a field would be nice. Any clues how I might go
>> about it?
>
> I used this code to get something working in just a few minutes.
> These objects can have superclasses, of course, and I am not
> checking for that. You could use Obj_Class to find all the
> superclass structures in an iterative way. The code would not
> be too hard to implement.
>
> The way this code works is to use Obj_Class to get the
> name of the object, which is the name of the structure
> you want. I create the right kind of structure with
> the Execute command. To extract the field, I have to
> use a second Execute statement.
>
> FUNCTION OBJECT::EXTRACT, field
>
> ; Check if FIELD is a valid field name. If it is,
> ; return the field. If not, return -1.

```
>
> thisClass = Obj_Class(self)
> ok = Execute('thisStruct = {' + thisClass + '}')
> structFields = Tag_Names(thisStruct)
> index = WHERE(structFields EQ StrUpCase(field), count)
> IF count EQ 1 THEN BEGIN
>   ok = Execute('Return, SELF.' + structFields(index(0)))
> ENDIF ELSE BEGIN
>   Message, 'Can not find field "' + field + $
>     '" in structure.', /Informational
>   RETURN, -1
> ENDELSE
> END
>
> This should give you some ideas.
>
> Cheers,
```

Looks good. I have one improvement, though. You can save yourself the second execute by using the handy field index trick, just like for structures.. i.e. just write:

```
return, SELF.(index[0])
```

Also, you needn't have the superclass info search, since the INHERITS definition in a class struct define simply includes all fields there.

JD
