Subject: Re: Function referencing/automatic defintion question.
Posted by David Fanning on Thu, 29 May 2003 15:14:39 GMT
View Forum Message <> Reply to Message

Paul van Delst (paul.vandelst@noaa.gov) writes:

> So my question is: what's the go here? Why doesn't my calling procedure "see" the compiled
> functions that precede my structure definition? I thought the whole point of sticking
> these routines *before* the procedure in my emiscoeff__define.pro file that actually does
> the definition meant that they would be compiled?
>
> Any insights appreciated,
>
> paulv
>
> p.s. When I manually compile the emiscoeff__define.pro file I get the following:
>
> IDL> .run emiscoeff__define
> % Compiled module: ASSOCIATED_EMISCOEFF.
> % Compiled module: DESTROY_EMISCOEFF.
> % Compiled module: ALLOCATE_EMISCOEFF.
> % Compiled module: ASSIGN_EMISCOEFF.
> % Compiled module: COUNT_EMISCOEFF_SENSORS.
> % Compiled module: EMISCOEFF__DEFINE.
>
> How come I don't get this list when I do the automatic compilation via
>
> EmisCoeff = { EmisCoeff }
>
> ???

Having the function in front of the object definition
module is a necessary, but not sufficient (at least in
this case) condition for getting it to compile correctly. :-)

The problem (almost certainly) is that a program
module that *calls* this function is being compiled
before the function is compiled.

You could solve this problem in several ways. (1) Take
the function out of this file and put it in a file of
its own. (2) Make the function a method of the object.

I think solution 2 is probably the better one in this case,
since the function is obviously related to the object in
a tight way. (In fact, I can't see why *all* of these modules
aren't object methods. Do you have a reason for this that is
not apparent to me?)

But if you want to keep it the way it is, I would just move
this function to the top of the file, or add a FORWARD_FUNCTION
statement in the module that uses it.

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Function referencing/automatic defintion question.
Posted by Paul Van Delst[1] on Thu, 29 May 2003 16:12:51 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Paul van Delst (paul.vandelst@noaa.gov) writes:
>
>> So my question is: what's the go here? Why doesn't my calling procedure "see" the compiled
>> functions that precede my structure definition? I thought the whole point of sticking
>> these routines *before* the procedure in my emiscoeff__define.pro file that actually does
>> the definition meant that they would be compiled?
>>
>> Any insights appreciated,
>>
>> paulv
>>
>> p.s. When I manually compile the emiscoeff__define.pro file I get the following:
>>
>> IDL> .run emiscoeff__define
>> % Compiled module: ASSOCIATED_EMISCOEFF.
>> % Compiled module: DESTROY_EMISCOEFF.
>> % Compiled module: ALLOCATE_EMISCOEFF.
>> % Compiled module: ASSIGN_EMISCOEFF.
>> % Compiled module: COUNT_EMISCOEFF_SENSORS.
>> % Compiled module: EMISCOEFF__DEFINE.
>>
>> How come I don't get this list when I do the automatic compilation via
>>
>> EmisCoeff = { EmisCoeff }
>>
>> ???

>
> Having the function in front of the object definition
> module is a necessary, but not sufficient (at least in
> this case) condition for getting it to compile correctly. :-)
>
> The problem (almost certainly) is that a program
> module that *calls* this function is being compiled
> before the function is compiled.

Umm...I'm not sure exactly what you mean. I have function,
Compute_Emissivity_Coefficients() that calls another function,
Compute_Theta_Coefficients(), which calls the EmisCoeff__Define procedure via the
structure definition,
  EmisCoeff = { EmisCoeff }
and then calls the Allocate_EmisCoeff() function (which resides in the
emiscoeff__define.pro source file *in front* of the EmisCoeff__Define procedure.

My apparently mistaken understanding is that the simple act of doing:
  EmisCoeff = { EmisCoeff }
will automatically compile Allocate_EmisCoeff() and make it available in the current scope
of the Compute_Theta_Coefficients() function (at the very least)

And, at the point where the function in question is called, it *has* already been
compiled. If I print out a list of the resolved functions *immediately* prior to the
Allocate_EmisCoeff function call, it's in the list:

```
IDL> .reset_session
IDL>  print, compute_emissivity_coefficients( 'test_sensor_emissivity.nc', EmisCoeff,
/pause)
% Compiled module: COMPUTE_EMISSIVITY_COEFFICIENTS.
% Compiled module: VALID_STRING.
% Compiled module: READ_NCDF.
% Compiled module: IS_NCDF.
% Compiled module: EMISCOEFF__DEFINE.
```

Printing the resolved function output from ROUTINE_INFO:

```
ALLOCATE_EMISCOEFF ASSIGN_EMISCOEFF ASSOCIATED_EMISCOEFF
CHECK_VECTORS
COMPUTE_EMISSIVITY_COEFFICIENTS COMPUTE_EMISSIVITY_FIT
COMPUTE_THETA_COEFFICIENTS
CONVERT_STRING
DESTROY_EMISCOEFF IS_NCDF MPCURVEFIT READ_NCDF SPLINE UNIQ VALID_STRING
% COMPUTE_THETA_COEFFICIENTS: Variable is undefined: ALLOCATE_EMISCOEFF.
% COMPUTE_EMISSIVITY_COEFFICIENTS: Error computing emissivity vs. theta fit coefficients.
       -1
IDL>
```

Note that ALLOCATE_EMISCOEFF is in the list.

> You could solve this problem in several ways. (1) Take
> the function out of this file and put it in a file of
> its own. (2) Make the function a method of the object.
>
> I think solution 2 is probably the better one in this case,
> since the function is obviously related to the object in
> a tight way. (In fact, I can't see why *all* of these modules
> aren't object methods. Do you have a reason for this that is
> not apparent to me?)

Because I don't want this project to descend into a object programming exercise. I like data encapsulation, but data hiding that requires get and set functions is just too much overhead for what I want to do (to say nothing of the terribly confusing [to me at least] syntax that uses "->"). From my point of view my named structure EmisCoeff *is* an "object". But it has public, rather than private, components.

At any rate, I just want to get my numbers and write them to a file so I can use my Fortran code to do something useful. The worst thing I did here was go from doing an "inline" structure definition to (what I thought would be) the more natty method of automatic structure defn.

> But if you want to keep it the way it is, I would just move
> this function to the top of the file, or add a FORWARD_FUNCTION
> statement in the module that uses it.

Thanks very much for the FORWARD_FUNCTION tip. That worked....but I don't understand in the least why it should be necessary.

cheers,

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7748
Fax:(301)763-8545

Subject: Re: Function referencing/automatic defintion question.
Posted by rmoss4 on Thu, 29 May 2003 16:52:15 GMT
View Forum Message <> Reply to Message

Paul van Delst wrote:
>

> Thanks very much for the FORWARD_FUNCTION tip. That worked....but I
> don't understand in the least why it should be necessary.
>
> cheers,
>
> paulv
>
>

I have run across the same problem, Paul, and I agree that it should not
be a problem. I have gotten in the habit of using

COMPILE_OPT IDL2

in virtually all of my programs. Since this forces one to properly use
brackets for array indices and parentheses for function calls, it
obviates the need for the use of FORWARD_FUNCTION. I'd be curious to
know if this solution would also solve your problem... I suspect it would.

--
Robert M. Moss, PhD

Subject: Re: Function referencing/automatic defintion question.
Posted by Paul Van Delst[1] on Thu, 29 May 2003 18:26:50 GMT
View Forum Message <> Reply to Message

Robert Moss wrote:
>
> Paul van Delst wrote:
>>
>> Thanks very much for the FORWARD_FUNCTION tip. That worked....but I
>> don't understand in the least why it should be necessary.
>>
>> cheers,
>>
>> paulv
>>
>>
>
> I have run across the same problem, Paul, and I agree that it should not
> *be* a problem. I have gotten in the habit of using
>
> COMPILE_OPT IDL2
>
> in virtually all of my programs. Since this forces one to properly use
> brackets for array indices and parentheses for function calls, it
> obviates the need for the use of FORWARD_FUNCTION. I'd be curious to

> know if this solution would also solve your problem... I suspect it would.

Yep, you're right. I removed the FORWARD_FUNCTION statement and replaced it with a COMPILE_OPT STRICTARR and everything worked fine. Thanks very much.

What this tells me is that the "default action" for IDL in this case is to assume that my function call is really an array operation where I'm using () instead of [] to subscript the array despite the fact that a function with the same name is compiled and resolved in the current scope. Huh?

This totally bamboozles me since I have a load of other source code files (including the main file for this little project) that have more than one pro/function in them (e.g. widget code with all the event handlers up front) with the "main" routine at the end. This is the *only* time I've ever had problems. My assumption that the compilation of automatic structure definition source files (the XXX__define type) is handled in the same way as other multi-pro/function source files is apparently wrong. If so, I wonder what bright spark decided that that would be a good idea?

cheers,

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7748
Fax:(301)763-8545

Subject: Re: Function referencing/automatic defintion question.
Posted by David Fanning on Thu, 29 May 2003 19:04:55 GMT
View Forum Message <> Reply to Message

Paul van Delst (paul.vandelst@noaa.gov) writes:

> What this tells me is that the "default action" for IDL in this case is to assume that my
> function call is really an array operation where I'm using () instead of [] to subscript
> the array despite the fact that a function with the same name is compiled and resolved in
> the current scope. Huh?
>
> This totally bamboozles me since I have a load of other source code files (including the
> main file for this little project) that have more than one pro/function in them (e.g.
> widget code with all the event handlers up front) with the "main" routine at the end. This
> is the *only* time I've ever had problems. My assumption that the compilation of automatic
> structure definition source files (the XXX__define type) is handled in the same way as
> other multi-pro/function source files is apparently wrong. If so, I wonder what bright
> spark decided that that would be a good idea?

Well, I'm a little confused, too. But I've been burned too
many times to jump on the bash the good folks at RSI
bandwagon just yet. Most of the time I end up finding
something stupid in my own code. :-(

I'm not sure the notion of "compiled and resolved
in the current scope" is terribly helpful. (For one
thing, I don't even know what it means.) My understanding
of the IDL compiler is that when it encounters an unresolved
token it checks (1) to see if something by that name
is already compiled and saved in the IDL code area,
(2) for a *.sav file with the same name as the token, then
(3) for a *.pro file with the same name as the token. Failing
all this, IDL gives you the benefit of the doubt
and assigns the token to its variable list.

It certainly isn't going to satisfy 2 or 3, so we have to
assume it is not on the "compiled already" list at the
time it checks the token. The real question is "Why not?"

Given the convoluted way this function was called, and
the sort of one-thing-after-another way computer software
is written, I think a plausible explanation might be that
even though IDL has compiled the function, it hasn't yet
had a chance to write the function on its function list,
so that at the time the list is *checked*, it is not there.

I don't feel confident enough about this to bet the ranch,
but I would wager a beer or two that the answer turns out
to be something like this. :-)

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: Function referencing/automatic defintion question.
Posted by Paul Van Delst[1] on Thu, 29 May 2003 19:40:32 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>

> Paul van Delst (paul.vandelst@noaa.gov) writes:
>
>> What this tells me is that the "default action" for IDL in this case is to assume that my
>> function call is really an array operation where I'm using () instead of [] to subscript
>> the array despite the fact that a function with the same name is compiled and resolved in
>> the current scope. Huh?
>>
>> This totally bamboozles me since I have a load of other source code files (including the
>> main file for this little project) that have more than one pro/function in them (e.g.
>> widget code with all the event handlers up front) with the "main" routine at the end. This
>> is the *only* time I've ever had problems. My assumption that the compilation of automatic
>> structure definition source files (the XXX__define type) is handled in the same way as
>> other multi-pro/function source files is apparently wrong. If so, I wonder what bright
>> spark decided that that would be a good idea?
>
> Well, I'm a little confused, too. But I've been burned too
> many times to jump on the bash the good folks at RSI
> bandwagon just yet. Most of the time I end up finding
> something stupid in my own code. :-(

Oh yeah - I've been there and done that plenty of times myself. It's the "good judgment comes from experience; experience comes from bad judgment" type of thing. :o) But you're right about the invective - I should replace the feet in my mouth (and count to 100 before posting stuff).

> I'm not sure the notion of "compiled and resolved
> in the current scope" is terribly helpful. (For one
> thing, I don't even know what it means.) My understanding
> of the IDL compiler is that when it encounters an unresolved
> token it checks (1) to see if something by that name
> is already compiled and saved in the IDL code area,

This is the case. #1

<snip>

> Given the convoluted way this function was called, and

Convoluted? If this function calling method is convoluted, then doesn't that say the same for object methods too? Don't get me wrong, I'm as adept at writing convoluted code as the next person, but this seems quite bone simple to me.

> the sort of one-thing-after-another way computer software
> is written, I think a plausible explanation might be that
> even though IDL has compiled the function, it hasn't yet
> had a chance to write the function on its function list,
> so that at the time the list is *checked*, it is not there.

But it is. If I print out the list of compiled functions immediately prior to the function call itself, the function in question is on it. The next line -- the function call -- fails because IDL thinks the function call is an array reference.

If the result of
  PRINT, ROUTINE_INFO(/FUNCTIONS)
contains the name of the function in question, then it has been compiled, right? And the fact that using
  COMILE_OPT STRICTARR
makes eveything work means the same thing. (Right? I think so.)

> I don't feel confident enough about this to bet the ranch,
> but I would wager a beer or two that the answer turns out
> to be something like this. :-)

Well the whole thing has got me beat.

cheers,

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7748
Fax:(301)763-8545

Subject: Re: Function referencing/automatic defintion question.
Posted by JD Smith on Fri, 30 May 2003 07:41:21 GMT
View Forum Message <> Reply to Message

On Thu, 29 May 2003 12:40:32 -0700, Paul van Delst wrote:

> If the result of
>   PRINT, ROUTINE_INFO(/FUNCTIONS)
> contains the name of the function in question, then it has been
> compiled, right? And the fact that using
>   COMILE_OPT STRICTARR
> makes eveything work means the same thing. (Right? I think so.)

There are routines which IDL knows about but hasn't compiled. They're called "unresolved". Try this:

 print,routine_info(/FUNCTIONS,/SOURCE)

Anything listed without source is unresolved. When IDL compiles files, it

records any procedures or functions it finds there, and only later actually goes looking for them. So, just being listed in routine info doesn't indicate a routine has been compiled (or even that it exists). Try compiling a file with a=mycrazyfunctionwhichwillneverexist() and you'll see it nonetheless.

I don't think there's anything wrong with your setup. I can put:

```
function stfunction,a
  return,a^2
end

pro stprocedure,b
  return
end

pro st__define,a
  a={ST,b:0}
end
```

in st__define.pro, and then:

```
IDL> a={st}
```

compiles the listed procedure and function by side-effect, and they work fine. The place this technique can go quite wrong, as it can for objects, is if the structure in question is already defined by some other means. Then IDL does not feel compelled to compile your __define fine, and your utility routines remain hidden. Any chance you use a full structure definition in creating a struct of this type anywhere else? You obviously can't mix the two methods for this to work.

JD

Subject: Re: Function referencing/automatic defintion question.
Posted by Paul Van Delst[1] on Fri, 30 May 2003 13:49:04 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
>
> On Thu, 29 May 2003 12:40:32 -0700, Paul van Delst wrote:
>
>> If the result of
>> PRINT, ROUTINE_INFO(/FUNCTIONS)
>> contains the name of the function in question, then it has been
>> compiled, right? And the fact that using

>>   COMILE_OPT STRICTARR
>> makes eveything work means the same thing. (Right? I think so.)
>
> There are routines which IDL knows about but hasn't compiled. They're
> called "unresolved".

That's what I meant in my previosu posts when I said they were compiled and resolved.

>   Try this:
>
>   print,routine_info(/FUNCTIONS,/SOURCE)

Here is the snippet of code where the weird stuff occurs (I commented out the COMPILE STRICTARR statement):

```
  ; ------------------------------------------
  ; Create and allocate the coefficient structure
  ; ------------------------------------------
  ; -- Create the structure
  EmisCoeff = { EmisCoeff }

  print,routine_info(/FUNCTIONS,/SOURCE)
  stop ;***NOTE the STOP ***
  ; -- Allocate it
  Result = Allocate_EmisCoeff( n_Wind_Speeds, n_Coefficients, n_Channels, $
                               EmisCoeff )
  IF ( Result NE SUCCESS ) THEN $
    MESSAGE, 'Error allocating EmisCoeff structure', $
             /NONAME, /NOPRINT
```

The output upon execution is (I put each listing of the ROUTINE_INFO output on a separate line so it's readable):

```
IDL> .reset_session
IDL>  print, compute_emissivity_coefficients( 'airsM9_aqua.SensorEmissivity.nc',
EmisCoeff, /pause)
% Compiled module: COMPUTE_EMISSIVITY_COEFFICIENTS.
% Compiled module: VALID_STRING.
% Compiled module: READ_NCDF.
% Compiled module: IS_NCDF.
% Compiled module: EMISCOEFF__DEFINE.
{ ALLOCATE_EMISCOEFF
 /usr2/wd20pd/idl/Emissivity/Sensor_Emissivity_Model/emiscoef f__define.pro}
{ASSIGN_EMISCOEFF
 /usr2/wd20pd/idl/Emissivity/Sensor_Emissivity_Model/emiscoef f__define.pro}
{ASSOCIATED_EMISCOEFF
 /usr2/wd20pd/idl/Emissivity/Sensor_Emissivity_Model/emiscoef f__define.pro}
{ COMPUTE_EMISSIVITY_COEFFICIENTS
 /usr2/wd20pd/f90/Emissivity/Sensor_Emissivity_Model/Regress_
```

Sensor_Emissivity/compute_emissivity_coefficients.pro}
{ COMPUTE_EMISSIVITY_FIT
 /usr2/wd20pd/f90/Emissivity/Sensor_Emissivity_Model/Regress_
Sensor_Emissivity/compute_emissivity_coefficients.pro}
{ COMPUTE_THETA_COEFFICIENTS
 /usr2/wd20pd/f90/Emissivity/Sensor_Emissivity_Model/Regress_
Sensor_Emissivity/compute_emissivity_coefficients.pro}
{ DESTROY_EMISCOEFF
 /usr2/wd20pd/idl/Emissivity/Sensor_Emissivity_Model/emiscoef f__define.pro}
{ MPCURVEFIT }
{SPLINE }
{ UNIQ }
% Stop encountered: COMPUTE_THETA_COEFFICIENTS 204
/usr2/wd20pd/f90/Emissivity/Sensor_Emissivity_M
   odel/Regress_Sensor_Emissivity/compute_emissivity_coefficien ts.pro

So, immediately before the call to Allocate_EmisCoeff(), it is in the compiled function
list, along with it's source file. I then type .cont:

IDL> .cont
% COMPUTE_THETA_COEFFICIENTS: Variable is undefined: ALLOCATE_EMISCOEFF.
% COMPUTE_EMISSIVITY_COEFFICIENTS: Error computing emissivity vs. theta fit coefficients.
         -1

> I don't think there's anything wrong with your setup. I can put:
>
> function stfunction,a
>   return,a^2
> end
>
> pro stprocedure,b
>   return
> end
>
> pro st__define,a
>   a={ST,b:0}
> end
>
> in st__define.pro, and then:
>
> IDL> a={st}
>
> compiles the listed procedure and function by side-effect, and they work
> fine. The place this technique can go quite wrong, as it can for objects,
> is if the structure in question is already defined by some other means.

It isn't. This is the *only* place in the code where I do
  EmisCoeff = { EmisCoeff }

I never create named structures for in-line structure definitions since I am always adding/changing stuff to/in them. But using the __define method creates a named structure so I know where it's happening. If I understand it correctly, a named structure is a different beastie from an unnamed one so it should be quite easy to tell the difference.

And when it reaches this part of the code I always get a

% Compiled module: EMISCOEFF__DEFINE.

message since all my tests have been preceded by a ".reset_session" to ensure I'm not shooting myself in the foot.

> Then IDL does not feel compelled to compile your __define fine, and your
> utility routines remain hidden. Any chance you use a full structure
> definition in creating a struct of this type anywhere else?

No, unfortunately (I say unfortunately because if my brainfade was the cause, that would make me happy).

> You obviously can't mix the two methods for this to work.

O.k. I can see that. But, again, the fact that everything works when I stick in a COMPILE_OPT STRICTARR statement suggests that a hidden, in-line definition that I forgot about somewhere else is not the problem.

Thanks for the info re: the ROUTINE_INFO. Although I'm even more bamboozled now - everything tells me the function is all ready to be used but IDL keeps thinking the call is an array reference rather than a function call.

cheers,

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7748
Fax:(301)763-8545

Subject: Re: Function referencing/automatic defintion question.
Posted by Robert.S.Hill.1 on Fri, 30 May 2003 14:12:20 GMT
View Forum Message <> Reply to Message

Paul van Delst <paul.vandelst@noaa.gov> writes:
> Here is the snippet of code where the weird stuff occurs (I commented
> out the COMPILE STRICTARR statement):

```
>
>    ; ---------------------------------------------
>    ; Create and allocate the coefficient structure
>    ; ---------------------------------------------
>    ; -- Create the structure
>    EmisCoeff = { EmisCoeff }
>
>    print,routine_info(/FUNCTIONS,/SOURCE)
>    stop ;***NOTE the STOP ***
>    ; -- Allocate it
>    Result = Allocate_EmisCoeff( n_Wind_Speeds, n_Coefficients,
>  n_Channels, $
>                                 EmisCoeff )
>    IF ( Result NE SUCCESS ) THEN $
>      MESSAGE, 'Error allocating EmisCoeff structure', $
>               /NONAME, /NOPRINT
```

I interject myself into this discussion with great trepidation, but...
Isn't this a compile-time issue, rather than a run-time issue? Unless Allocate_EmisCoeff has been compiled already when the routine containing the snippet is compiled, then Allocate_EmisCoeff is going to look like a variable no matter what the situation is at run time. Or did I miss something in the previous discussion, in which case I am baffled, too. But this is precisely the sort of thing that forward_function takes care of.

-Bob Hill

Subject: Re: Function referencing/automatic defintion question.
Posted by Paul Van Delst[1] on Fri, 30 May 2003 14:47:04 GMT
View Forum Message <> Reply to Message

"Robert S. Hill" wrote:

```
>
>  Paul van Delst <paul.vandelst@noaa.gov> writes:
>>  Here is the snippet of code where the weird stuff occurs (I commented
>>  out the COMPILE STRICTARR statement):
>>
>>    ; ---------------------------------------------
>>    ; Create and allocate the coefficient structure
>>    ; ---------------------------------------------
>>    ; -- Create the structure
>>    EmisCoeff = { EmisCoeff }
>>
>>    print,routine_info(/FUNCTIONS,/SOURCE)
>>    stop ;***NOTE the STOP ***
>>    ; -- Allocate it
```

>>   Result = Allocate_EmisCoeff(n_Wind_Speeds, n_Coefficients,
>> n_Channels, $
>>   EmisCoeff)
>>   IF (Result NE SUCCESS) THEN $
>>   MESSAGE, 'Error allocating EmisCoeff structure', $
>>   /NONAME, /NOPRINT
>
> I interject myself into this discussion with great trepidation, but...

Please...interject. This is driving me nuts (can't you tell :o)

> Isn't this a compile-time issue, rather than a run-time issue? Unless
> Allocate_EmisCoeff has been compiled already when the routine containing
> the snippet is compiled, then Allocate_EmisCoeff is going to look like a
> variable no matter what the situation is at run time.

Well, if this is true, then I throw my hands up. (In a good way :o) My bread-n-butter is writing Fortran90/95 code so for me (via that context) compile-time and run-time mean very specific things. To be honest -- and here I expose my ignorance -- I don't separate compile- and run-time in IDL (at least how I understand it).

My understanding is that when I _run_ the routine containing the snippet above it gets to the line where the structure is defined and _compiles_ all the routines in the source file emiscoeff__define.pro. After that my assumption is that all of those emiscoeff__define.pro contained routines are available for use in the current scope, i.e. in the routine that calls Allocate_EmisCoeff().

If I understand you correctly, this is not the case?

And, even if it is, that doesn't explain why the use of COMPILE_OPT STRICTARR makes everything work as I would expect. It's as if IDL treats an array reference with higher precedence than a function call when it's given a choice.

Thanks. Your interjection is a pillow between my head and the brick wall in front of me. :o) (I think I'm becoming compulsive-obsessive, but I just _have_ to figure this out.)

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7748
Fax:(301)763-8545

Subject: Re: Function referencing/automatic defintion question.
Posted by Robert.S.Hill.1 on Fri, 30 May 2003 15:22:07 GMT

Paul van Delst <paul.vandelst@noaa.gov> writes:
> Please...interject. This is driving me nuts (can't you tell :o)

Well, then -- emboldened, I press on.

> My understanding is that when I _run_ the routine containing the snippet
> above it gets to the line where the structure is defined and _compiles_
> all the routines in the source file emiscoeff__define.pro. After that
> my assumption is that all of those emiscoeff__define.pro contained
> routines are available for use in the current scope, i.e. in the
> routine that calls Allocate_EmisCoeff().

Just to be clear, here is more detail on what I think is probably happening. I'm assuming here that your calling code is from a main level program that you run using the .run command.

The .run command doesn't interpret your calling program line by line. Instead, it compiles it all into bytecode (or whatever RSI calls it), then executes it. During this execution, the execution engine arrives at your structure invocation with the curly braces, and it then invokes the compiler to compile all the routines in the __define file. Subsequently, the execution engine reaches the Allocate_EmisCoeff() invocation, but this has *already* been compiled as an array, so it doesn't recognize it as a function (an array and function of the same name can coexist happily).

Although compile time and run time are not globally separated as in Fortran or C, they are separate for each routine, including any main level script. Even when you put a bunch of routines in one file, you need to be aware of the dependence hierarchy of any of them that are functions, and put the inner ones higher up in the file. (Or use strictarr or forward_function.)

-Bob Hill

Subject: Re: Function referencing/automatic defintion question.
Posted by Paul Van Delst[1] on Fri, 30 May 2003 16:29:41 GMT

"Robert S. Hill" wrote:
>
> Paul van Delst <paul.vandelst@noaa.gov> writes:
>> Please...interject. This is driving me nuts (can't you tell :o)
>
> Well, then -- emboldened, I press on.

>
>> My understanding is that when I _run_ the routine containing the snippet
>> above it gets to the line where the structure is defined and _compiles_
>> all the routines in the source file emiscoeff__define.pro. After that
>> my assumption is that all of those emiscoeff__define.pro contained
>> routines are available for use in the current scope, i.e. in the
>> routine that calls Allocate_EmisCoeff().
>
> Just to be clear, here is more detail on what I think is probably
> happening. I'm assuming here that your calling code is from a main
> level program that you run using the .run command.

Nope. I never do that. My calling code is itself a function that I invoke from the main level thusly:

IDL> print, compute_emissivity_coefficients('test_sensor_emissivity.nc', EmisCoeff)

> The .run command doesn't interpret your calling program line by line.
> Instead, it compiles it all into bytecode (or whatever RSI calls it),
> then executes it. During this execution, the execution engine arrives
> at your structure invocation with the curly braces, and it then invokes
> the compiler to compile all the routines in the __define file.
> Subsequently, the execution engine reaches the Allocate_EmisCoeff()
> invocation, but this has *already* been compiled as an array, so it
> doesn't recognize it as a function (an array and function of the same
> name can coexist happily).

I'm thinking that something like this is happening but I don't understand exactly why. My assumption has always been that once a routine has been compiled by default (i.e. it precedes the routine that a source code file is named after in the file - your "inner" routine) then that routine is accessible in all and any subsequent procedure/function independent of their heirarchy. I think that assumption is flawed. My working assumption now is that the "inner" routines in an IDL source code file are really only accessible to the "outer" routine in a source file (i.e. with the same name as the file itself.) *unless* you do something like the compile_opt strictarr or forward_function thingo.

> Although compile time and run time are not globally separated as in
> Fortran or C, they are separate for each routine, including any main
> level script. Even when you put a bunch of routines in one file, you
> need to be aware of the dependence hierarchy of any of them that are
> functions, and put the inner ones higher up in the file.

This I religiously do so...

> (Or use strictarr or forward_function.)

I've never needed these before.

The germ of a resolution is forming in my mind. I'm going to test some stuff when I get some time next week. Thanks very much.

When I finally figure this out I just *know* everybody else will say "Well,...yeah, of course - why would you think it would work the other way?" :o)

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC
Ph: (301)763-8000 x7748
Fax:(301)763-8545

Subject: Re: Function referencing/automatic defintion question.
Posted by David Fanning on Fri, 30 May 2003 16:50:32 GMT
View Forum Message <> Reply to Message

Paul van Delst (paul.vandelst@noaa.gov) writes:

> When I finally figure this out I just *know* everybody else will say "Well,...yeah, of
> course - why would you think it would work the other way?" :o)

I'm already thinking that, but--of course--I haven't
seen the solution. :-)

Cheers,

David

--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155
