Subject: Re: overload init function in class/object ?
Posted by David Fanning on Tue, 06 May 2003 19:48:22 GMT
View Forum Message <> Reply to Message

paul wisehart (wisehart@runbox.com) writes:

> I was under the assumption that you could overload the
> init function of an object.
>
> However, it doesn't work for me.
>
> In the below example I get an error if I try:
> IDL>o = obj_new('obj')
>
> Yet this works:
> IDL>o = obj_new('obj','test')
>
> I wanna be able to overload my init functions!
> Please tell me I'm doing something wrong.
>
>
> ;--- obj__define.pro --------------------
>
> function obj::init
> compile_opt idl2
> return, 1
> end
> ;-----------------------
> function obj::init, p1
> compile_opt idl2
> print, p1
> return, 1
> end

I think it is your definition of "overload" that
is probably doing you in. :-)

I'm not sure what you mean by it, but it sure doesn't
mean "define the same thing twice, in two different ways".

What exactly are you trying to do here?

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: overload init function in class/object ?
Posted by Mark Hadfield on Tue, 06 May 2003 19:55:37 GMT
View Forum Message <> Reply to Message

"paul wisehart" <wisehart@runbox.com> wrote in message
news:slrnbbg37q.2t7.wisehart@stingray.mcst.ssai.biz...
> I was under the assumption that you could overload the
> init function of an object.
>
> However, it doesn't work for me.
>
> In the below example I get an error if I try:
> IDL>o = obj_new('obj')
>
> Yet this works:
> IDL>o = obj_new('obj','test')
>
> I wanna be able to overload my init functions!
> Please tell me I'm doing something wrong.
>
>
> ;--- obj__define.pro --------------------
>
> function obj::init
> compile_opt idl2
> return, 1
> end
> ;-----------------------
> function obj::init, p1
> compile_opt idl2
> print, p1
> return, 1
> end
> ...

You can't overload a function in this way in IDL. When IDL compiles your obj__define.pro the second version of obj::init will *replace* the first. However IDL allows optional parameters, so the way to handle this in IDL is something like

function obj::init, p1
  compile_opt idl2

```
   if n_elements(p1) gt 0 then print, p1
   return, 1
end
```

The exact form of the test for the presence of the p1 positional parameter depends on what you want to do with it; "if n_elements(p1) gt 0" is generally what you want for input parameters.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: overload init function in class/object ?
Posted by paul wisehart on Tue, 06 May 2003 20:14:12 GMT
View Forum Message <> Reply to Message

On Wed, 7 May 2003 07:55:37 +1200, Mark Hadfield <m.hadfield@niwa.co.nz> wrote:
> You can't overload a function in this way in IDL. When IDL compiles your
> obj__define.pro the second version of obj::init will *replace* the first.
> However IDL allows optional parameters, so the way to handle this in IDL is
> something like

I knew I was missing something!
When you put it that way my mistake(and solution) is obvious (to me).
I still have lingering C++ syntax floating around my brain.

Thanks alot!

(So technically speaking you _cannot_ overload functions in
IDL? You just use optional parameters to get the same effact?)

--
paul \ /
wisehart >/
<//////$>
|\|\|\

Subject: Re: overload init function in class/object ?
Posted by paul wisehart on Tue, 06 May 2003 20:20:07 GMT
View Forum Message <> Reply to Message

On Tue, 6 May 2003 13:48:22 -0600, David Fanning <david@dfanning.com> wrote:
> paul wisehart (wisehart@runbox.com) writes:
> I think it is your definition of "overload" that

> is probably doing you in. :-)
>
> I'm not sure what you mean by it, but it sure doesn't
> mean "define the same thing twice, in two different ways".
In C++ thats exectly what it means :)
Furthermore, thats what the term "function overloading"
means in general. But, I see know that I was incorrect about IDL's
interpretation of "function overloading"

--
paul \ /
wisehart >/
<//////$>
|\|\|\

Subject: Re: overload init function in class/object ?
Posted by David Fanning on Tue, 06 May 2003 20:26:13 GMT
View Forum Message <> Reply to Message

paul wisehart (wisehart@runbox.com) writes:

> (So technically speaking you _cannot_ overload functions in
> IDL? You just use optional parameters to get the same effact?)

Isn't that a distinction without a difference?

When your parameters can be anything you would like
them to be, and you can have as many as you like,
what else could possibly be meant by "overloading".

Here is code for a GETCOLOR method on my ColorTool object.
The variable theColor can be absent, a scalar string, or a
string array. The variable colorIndex can be absent, a
scalar, or an array.

```
FUNCTION ColorTool::GetColor, theColor, colorIndex, $
   AllColors=allcolors, $
   Cancel=cancelled, $
   Decomposed=decomposedState, $
   _Extra=extra, $
   Names=names, $
   Row=row, $
   Triple=triple, $
   _Extra=extraKeywords

   @cat_error_handler
```

```
; Make sure you have a color name and color index.

CASE N_Elements(theColor) OF
  0: BEGIN
    theColor = 'White'
    IF N_Elements(colorIndex) EQ 0 THEN $
      colorIndex = !P.Color < (!D.Table_Size - 1) $
      ELSE colorIndex = 0 > colorIndex < (!D.Table_Size - 1)
    ENDCASE

  1: BEGIN
      type = Size(theColor, /TNAME)
      IF type NE 'STRING' THEN $
        Message, 'The color must be expressed as a color name.'
      IF N_Elements(colorIndex) EQ 0 THEN $
        colorIndex = !P.Color < (!D.Table_Size - 1) $
        ELSE colorIndex = 0 > colorIndex < (!D.Table_Size - 1)
      ENDCASE

  ELSE: BEGIN
      type = Size(theColor, /TNAME)
      IF type NE 'STRING' THEN $
        Message, 'The colors must be expressed as color names.'
      ncolors = N_Elements(theColor)
      CASE N_Elements(colorIndex) OF
        0: colorIndex = Indgen(ncolors) + $
             (!D.Table_Size - (ncolors + 1))
        1: colorIndex = Indgen(ncolors) + colorIndex
        ELSE: IF N_Elements(colorIndex) NE ncolors THEN $
          Message, 'Index vector must be the same ' + $
                'length as color name vector.'
      ENDCASE

  ENDCASE
```

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: overload init function in class/object ?
Posted by David Fanning on Tue, 06 May 2003 20:34:06 GMT
View Forum Message <> Reply to Message

paul wisehart (wisehart@runbox.com) writes:

>> I'm not sure what you mean by it, but it sure doesn't
>> mean "define the same thing twice, in two different ways".
> In C++ thats exectly what it means :)
> Furthermore, thats what the term "function overloading"
> means in general. But, I see know that I was incorrect about IDL's
> interpretation of "function overloading"

Really!? I can see now why we are *both* confused, because that
sure as hell doesn't make any sense to me! :-)

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: overload init function in class/object ?
Posted by Mark Hadfield on Tue, 06 May 2003 20:34:41 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.1921c851ec23ee94989b86@news.frii.com...
> paul wisehart (wisehart@runbox.com) writes:
>
>> (So technically speaking you _cannot_ overload functions in
>> IDL? You just use optional parameters to get the same effact?)
>
> Isn't that a distinction without a difference?

Not to someone who's used to C++. In that language you can have several versions of the same function, differing only in the number (and/or type?) of their arguments. The compiler examines the argument list at each place in the code where the function is called and dispatches the call to the appropriate version. Because the decision is made at compile time (C++ being a statically typed language) it is more efficient than doing run-time checks inside the function in the IDL manner. But less flexible.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: overload init function in class/object ?
Posted by David Fanning on Tue, 06 May 2003 20:38:38 GMT
View Forum Message <> Reply to Message

Mark Hadfield (m.hadfield@niwa.co.nz) writes:

> Not to someone who's used to C++. In that language you can have several
> versions of the same function, differing only in the number (and/or type?)
> of their arguments. The compiler examines the argument list at each place in
> the code where the function is called and dispatches the call to the
> appropriate version. Because the decision is made at compile time (C++ being
> a statically typed language) it is more efficient than doing run-time checks
> inside the function in the IDL manner. But less flexible.

OK, I had "learn C++" on my to-do list, but I've
taken it off. I'm joining up with the object Luddites
if that's the kind of nonsense I have to learn! :-(

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: overload init function in class/object ?
Posted by Mark Hadfield on Tue, 06 May 2003 20:55:02 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.1921cb3ca539d5c989b89@news.frii.com...
> Mark Hadfield (m.hadfield@niwa.co.nz) writes:
>> [Description of function overloading in C++ omitted.]
>
> OK, I had "learn C++" on my to-do list, but I've

> taken it off. I'm joining up with the object Luddites
> if that's the kind of nonsense I have to learn! :-(

That's not the half of it. Wait till you get to virtual base classes!

But you might want to add "fool around with Python" to your to-do list.
Python is a dynamically typed language, much more similar in spirit to IDL
but with a more (shall we say) *coherent* design. And the entry barrier is
low: installing the Python interpreter and writing a little script to
process some text should take only an hour or two.

--

Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: overload init function in class/object ?
Posted by James Kuyper on Tue, 06 May 2003 21:29:24 GMT
View Forum Message <> Reply to Message

paul wisehart wrote:
>
> On Tue, 6 May 2003 13:48:22 -0600, David Fanning <david@dfanning.com> wrote:
>> paul wisehart (wisehart@runbox.com) writes:
>> I think it is your definition of "overload" that
>> is probably doing you in. :-)
>>
>> I'm not sure what you mean by it, but it sure doesn't
>> mean "define the same thing twice, in two different ways".
> In C++ thats exectly what it means :)

Well, not precisely. In C++, overloading means defining two different
things with the same exact name, in possibly different ways, and letting
the compiler decide which one to use, based upon the number and types of
the arguments. Given the different way that IDL handles data types, and
the different way it handles default arguments, it's neither feasible
nor necessary to use C++ type overloading in IDL.

Subject: Re: overload init function in class/object ?
Posted by JD Smith on Tue, 06 May 2003 21:32:45 GMT
View Forum Message <> Reply to Message

On Tue, 06 May 2003 13:34:06 -0700, David Fanning wrote:

> paul wisehart (wisehart@runbox.com) writes:
>
>>> I'm not sure what you mean by it, but it sure doesn't mean "define
>>> the same thing twice, in two different ways".
>> In C++ thats exectly what it means :) Furthermore, thats what the term
>> "function overloading" means in general. But, I see know that I was
>> incorrect about IDL's interpretation of "function overloading"
>
> Really!? I can see now why we are *both* confused, because that sure as
> hell doesn't make any sense to me! :-)
>

In statically types languages such as C++, and other optionally-typed languages, there is the notion of a subroutine's "signature", i.e. its return type, number of arguments, and each argument's type, taken together. Using the signature, a specific method can be chosen from among a number of methods with the same name, but differing signatures, e.g.:

```
class Example {
  // same name, three different methods
 int sum (int a) { return a; }
 int sum (int a, int b) { return a + b; }
 int sum (int a, int b, int c) { return a + b + c; }
}
```

Since such languages must figure out which version of the "sum" method to call (i.e. "bind" the method) at compile time, this is a nice way of preserving the semantics of an operation (like "sum") without resorting to ugly names like sum_one, sum_two, and sum_three. Since IDL is not statically-typed (and why would we want it to be), it has (almost) no notion of signature overloading, nor does it need it. The only situation where it does permit it relates to the duality of procedures vs. functions, which can be thought of as a minimal "binary" signature (e.g., returns nothing, or returns something). You are allowed to use a single name for both a procedure and function method within a class, e.g.:

```
pro Obj::Foo,f
  self.foo=f
end

function Obj::Foo
  return,self.foo
end
```

and IDL will happily distinguish

obj->Foo

from

foo=obj->Foo()

However, since arguments are always optional by default, and type is a run-time property, signature overloading is easily reproduced, and made even more convenient with keywords. Another bonus: you probably won't find yourself in need of signature overloading very often, since IDL's lack of type enforcement allows you to pass many more compatible data types into the same piece of code. Whereas C++ would need:

int sum(int a, int b)
float sum(float a, float b)
float *sum(float *a, float *b)
char *sum(char *a, char *b)
...

IDL can use the exact same method to add any numeric, array, or string data type.

Note that overloading contrasts markedly from "overriding", which IDL does support, and relates to the familiar parent/child method of replacement or refinement typified in statements like:

if (self->Parent::Init(_EXTRA=e) ne 1) then return,0

JD

Subject: Re: overload init function in class/object ?
Posted by paul wisehart on Wed, 07 May 2003 01:43:10 GMT
View Forum Message <> Reply to Message

This thread has been _extremely_ enlightening for me.
I really appreciate all of the input.

I am still new to IDL (pretty obvious from my questions eh? :)
I still see IDL as a non-native speaker sees a new language.
I subconsiously translate everything into what I know.
When someone gets to the point where they really _know_ a new language they stop translating(even if sub-consciously) to there native language and start thinking _directly_ in the new language.

Optional parameters is area of IDL where I am still very much

not a native speaker. In the code I write I force myself to explicitly use every parameter that I pass. Where more experienced IDL programmers use optional parameters I use extra sub-routines.

I am pretty sure this is an example of how I translate IDL into C++ sub-consciously.

Thanks all for pointing this out to me!

--
paul wisehart

(ps. eventually I will start using IDLWAVE, and my vi/vim trained fingers are trembling in fear ... boo!)
