Subject: taking the widget plunge. help
Posted by Peter Brooker on Wed, 06 Sep 2000 19:54:54 GMT
View Forum Message <> Reply to Message

I need to write a program to learn how easy/hard IDL widgets are to use. I figure that a starting point would be a well documented simple program that demostrates how they are to use.

Any suggestions?

thanks-Peter Brooker

Subject: Re: taking the widget plunge. help
Posted by Martin Schultz on Tue, 12 Sep 2000 08:32:04 GMT
View Forum Message <> Reply to Message

"J.D. Smith" wrote:
>
>> P.S. Let's just say I spent a couple of days writing
>> documentation for FSC_PSConfig, *the* best program I've
>> ever written, and no one downloads it or uses it. You just
>> figure after a while, what's the point?
>
> I took this as a friendly reminder to check out your new stuff, and I was
> frankly amazed at how much work you'd put into the documentation. Really an
> astounding effort. I will try to integrate FSC_PSConfig into some of my
> programs around here.
>
> One note which I think is instructive. You include a section on customization
> where you outline how to directly modify your source to add personal or
> company-wide set-up lists. This is a very useful feature, but I think you're
> going to cause yourself and potential users grief here. It's a *perfect* place
> to flex our object oriented muscles. The problem will be that in a year you'll
> think of a great way to redesign it, or maybe RSI makes some changes to device
> which prompt a rewrite. Then, either all the users who have made their own
> modifications will be out of luck, or you'll be contrained in what kind of
> updates you can do. It is exactly these types of situations that scream out for
> some sort of object relationship. If, rather than giving direction on how to
> change your code, you gave a simple example of INHERIT'ing your class, and
> chaining to its setup code, you could fully preserve "forward compatibility" --
> i.e. drop-in replacement of your updated code.
>
I fully agree. Isn't inheriting what objects are all about?

> Or, since in this case the local setup changes are data-only (no fundamental
> method changes), you could simply provide access to an internally growable list
> of setups. Inheritance is not even really required.

>
> I haven't looked closely, but a method which allows you to add setup lists
> (e.g. self->AddSetup,"Company Viewgraph",/EUROPEAN, FontNameSet="Helvetica"),
> would seem to do the trick. This might be called automatically in Init with all
> the built-in defaults. A user could INHERIT it, override and chain to AddSetup
> for a fully internal solution, or they could use a compound relationship and add
> the setups "from the outside" in whatever wrapper routine (or object) they
> use.
>
> The details of how set-ups are stored, manipulated, etc., would be hidden, only
> the published interface of AddSetup need remain the same (or backwards
> compatible anyway... nothing to stop you adding new keywords as new features
> become available).
>
> Anyway, it's just a thought. Perhaps you're afraid of scaring off potential
> users with objects. You shouldn't be. It's good for them.
>
> JD

This seems somewhat "convoluted" to me (but after recent experience, I
am sure you will have your reasons for proposing exactly this). I
always tend to think that setup is best done with ASCII files that are
easily editable and human readable. Yes, you should have a method
named something like FSC_PsConfig::Setup, and this method should
define a minimal set of defaults. But then it would read a file and
overload the default definitions. If it doesn't find the file, well,
then you live with the defaults (or the company creates a child object
with specific defaults). Proposed strategy:

1. define a few defaults (IDL standard, letter portrait, letter
landscape)
2. look for setup file ('psconfig.setup') first in local directory,
then in IDL !PATH -
   potentially also in !FSC_CONFIG_PATH if it is defined
3. if found, analyze file and overrule default settings

Useful additional keywords to setup would be:
/Add (and name, size, ...) : add a specific setting "manually"
/Help : print out a help message describing the setup file format and
search method

As for the file format you could do something like
A4:
size=11.9,6.2   # not sure about the values
color=1
END

A4_Landscape:

size=6.2,11.9
color=0
END

etc.

Poor David! The more you do, the more you are asked to do...

Cheers,
Martin

PS: If you like a parser method for a format like this, I can send you
one.

--
[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[

Subject: Re: taking the widget plunge. help
Posted by John-David T. Smith on Tue, 12 Sep 2000 15:42:58 GMT
View Forum Message <> Reply to Message

Martin Schultz wrote:

>
> This seems somewhat "convoluted" to me (but after recent experience, I
> am sure you will have your reasons for proposing exactly this). I
> always tend to think that setup is best done with ASCII files that are
> easily editable and human readable. Yes, you should have a method
> named something like FSC_PsConfig::Setup, and this method should
> define a minimal set of defaults. But then it would read a file and
> overload the default definitions. If it doesn't find the file, well,
> then you live with the defaults (or the company creates a child object
> with specific defaults). Proposed strategy:
<snip>
> As for the file format you could do something like
> A4:
> size=11.9,6.2   # not sure about the values
> color=1
> END
>
> A4_Landscape:

> size=6.2,11.9
> color=0
> END
>

The problem with using a text file for the input, is that it's deceptively appealing. Easy to edit, no object knowledge required, etc. But, once you've set the format, you're basically locked into it. Want to add some new items or reorganize (for instance, making groups of setups)? You'll need special code to handle older-format input files (though you could obviously plan ahead for such contingencies). Want to reorganize the internal representation of the data entirely? You'll still have to accomodate the old input mechanism. For this problem, a flat-file input is probably tractable, but I thought it would be a good example case for maximizing forward compatibility. Backward compatibility is easy, if tedious. Forward compatibility (being able to replace aging modules/objects with new ones without changing the including code), is more troublesome.

The idea of abstracting the interface to be limited to a defined set of methods with given arguments contrains the fixed interface specification to elements enforced by the language itself... certainly RSI won't change the meaning of arguments or remove keyword functionality. This abstraction is certainly sometimes overkill, as it is not without its costs. But for something which is intended to be upgradeable and extensible, I think it can be worth it.

JD

--
J.D. Smith /*\ WORK: (607) 255-6263
Cornell University Dept. of Astronomy */ (607) 255-5842
304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
Ithaca, NY 14853 */

Subject: Re: taking the widget plunge. help
Posted by Martin Schultz on Tue, 12 Sep 2000 16:24:30 GMT
View Forum Message <> Reply to Message

"J.D. Smith" wrote:
>
> Martin Schultz wrote:
>
>>
>> This seems somewhat "convoluted" to me (but after recent experience, I
>> am sure you will have your reasons for proposing exactly this). I
>> always tend to think that setup is best done with ASCII files that are
>> easily editable and human readable. Yes, you should have a method
>> named something like FSC_PsConfig::Setup, and this method should

>> define a minimal set of defaults. But then it would read a file and
>> overload the default definitions. If it doesn't find the file, well,
>> then you live with the defaults (or the company creates a child object
>> with specific defaults). Proposed strategy:
> <snip>
>> As for the file format you could do something like
>> A4:
>> size=11.9,6.2 # not sure about the values
>> color=1
>> END
>>
>> A4_Landscape:
>> size=6.2,11.9
>> color=0
>> END
>>
>
> The problem with using a text file for the input, is that it's deceptively
> appealing. Easy to edit, no object knowledge required, etc. But, once you've
> set the format, you're basically locked into it. Want to add some new items or
> reorganize (for instance, making groups of setups)? You'll need special code to
> handle older-format input files (though you could obviously plan ahead for such
> contingencies). Want to reorganize the internal representation of the data
> entirely? You'll still have to accomodate the old input mechanism. For this
> problem, a flat-file input is probably tractable, but I thought it would be a
> good example case for maximizing forward compatibility. Backward compatibility
> is easy, if tedious. Forward compatibility (being able to replace aging
> modules/objects with new ones without changing the including code), is more
> troublesome.
>
> The idea of abstracting the interface to be limited to a defined set of methods
> with given arguments contrains the fixed interface specification to elements
> enforced by the language itself... certainly RSI won't change the meaning of
> arguments or remove keyword functionality. This abstraction is certainly
> sometimes overkill, as it is not without its costs. But for something which is
> intended to be upgradeable and extensible, I think it can be worth it.
>
> JD
>
> --
> J.D. Smith /*\ WORK: (607) 255-6263
> Cornell University Dept. of Astronomy */ (607) 255-5842
> 304 Space Sciences Bldg. /*\ FAX: (607) 255-5875
> Ithaca, NY 14853 */

I see. Yet, I would like to argue for text based setup files, because
they are relatively easy maintainable and human readable. If I compare
the Windows registry with the Unix ASCII based setup files, I

certainly prefer the latter (for example one can use grep to find something, and one can add comments). With a format like the one I indicated, you still guarantee a lot of up- and downward compatibility: Variables that are undefined in one version will simply produce a warning message but otherwise be ignored. BTW: I don't agree that backward compatibility is always easy - at least not, if you are dealing with binary files of any kind (including IDL sav files).

I do concede, however, that ASCII files bear the danger of getting messy and "incompatible". To some extent, this can be solved by setting some standards such as:

- comments begin with '#'
- definition sections begin with a name followed by ':' and end with 'END'
- a setup file contains either no definition sections (i.e. variables are defined directly or "globally") or only definition sections
- each variable definition must contain a '=' even if the definition is empty

and to facilitate the search for the file:

- the filename of the setup file is <programname>.setup

One advantage compared to the sort of "internal" setup you are proposing is that even people who don't know much about IDL can customize the program (errr, the object ;-) whereas you need to know how to inherit if you want to overwrite a setup method. Well, on the other hand, this would give even more power to skilled programmers who can get rich ...

Cheers,
Martin

--

[[[[[[[[[[
[[Dr. Martin Schultz   Max-Planck-Institut fuer Meteorologie   [[
[[Bundesstr. 55, 20146 Hamburg   [[
[[phone: +49 40 41173-308   [[
[[fax: +49 40 41173-298   [[
[[martin.schultz@dkrz.de   [[
[[[[[[[[[[

Subject: Re: taking the widget plunge. help
Posted by davidf on Wed, 13 Sep 2000 13:02:33 GMT
View Forum Message <> Reply to Message

I want to thank JD and Martin for organizing

my Fall programming schedule. I'm so busy working I really didn't have time to do it myself, so I appreciate it. :-)

JD writes about a program that is getting downloaded from my web page rather more often lately:

> One note which I think is instructive. You include a section on customization
> where you outline how to directly modify your source to add personal or
> company-wide set-up lists. This is a very useful feature, but I think you're
> going to cause yourself and potential users grief here. It's a *perfect* place
> to flex our object oriented muscles. The problem will be that in a year you'll
> think of a great way to redesign it, or maybe RSI makes some changes to device
> which prompt a rewrite. Then, either all the users who have made their own
> modifications will be out of luck, or you'll be contrained in what kind of
> updates you can do. It is exactly these types of situations that scream out for
> some sort of object relationship. If, rather than giving direction on how to
> change your code, you gave a simple example of INHERIT'ing your class, and
> chaining to its setup code, you could fully preserve "forward compatibility" --
> i.e. drop-in replacement of your updated code.

I actually did give this quite a bit of thought at the time I wrote the program. And I considered Martin's idea of text files, too. And for the same reason he likes them: they are easy for inexperienced users to figure out. I decided against the first option because I had run out of steam writing the documentation and I figured that if I had to write the documentation for object inheritance I might as well write the book I was writing the damn program to avoid writing. (If you know what I mean.)

I decided against text files, because I already had a problem with multiple files and I didn't really want to add any more.

But both of these ideas are worth considering, and if my ftp logs show that anyone besides JD and Martin are using the darn thing I may have to re-consider my earlier decisions. :-)

Cheers,

David

--
David Fanning, Ph.D.

Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155