Subject: Re: subverting IDL builtin variables !FORMYOWNPURPOSES
Posted by Jeremy Bailin on Sat, 19 Feb 2011 01:58:28 GMT
View Forum Message <> Reply to Message

defsysv?

-Jeremy.

Subject: Re: subverting IDL builtin variables !FORMYOWNPURPOSES
Posted by Michael Galloy on Sat, 19 Feb 2011 03:34:56 GMT
View Forum Message <> Reply to Message

On 2/18/11 5:41 PM, Ed Hyer wrote:
> This is a terrible idea, but I'll feel better about doing it one of
> the Hard Ways once I've committed this awful cheat to posterity.
>
> The package has ~100 subroutines, and each subroutine has various
> types of output. All of this output ends up in logs, but now that the
> whole creature is built, it's time to set different levels of
> verbosity. My simple scheme is like this:
>   DEBUG=0; put nothing in the log except fatal errors;
>   DEBUG=1; include warnings and limited diagnostics;
>   DEBUG=2; include full diagnostics, performance-related
> information, the kitchen sink.
>
> Now, I can think of three ways to do this:
> 1) Pass a VERBOSITY keyword from the top level through all of the
> subroutines. I'm not going to change 100 headers to add this (though I
> am going to change ~200 PRINT statements to IF(VERBOSITY gt XX) THEN
> PRINT).
> 2) Create a common block for the VERBOSITY level. I've never done
> this, but it seems like the right solution for this problem.
> 3) Put the VERBOSITY into a !VARIABLE that isn't being used for
> anything else. There are plenty to choose from, especially since this
> package doesn't actually generate any graphics, etc.
>
> Solution #3 is so easy... so wrong... so easy. Oh well.
> Have a great weekend, everybody.

I ended up using common blocks for my logging framework, but system
variables would work just as well.

See MG_LOG and MGffLogger in the dist_tools for the way I did it:

  http://docs.idldev.com/dist_tools/

By the way, you don't have to subvert a pre-existing system variable, you can create your own with DEFSYSV.

Mike
--
www.michaelgalloy.com
Research Mathematician
Tech-X Corporation

Subject: Re: subverting IDL builtin variables !FORMYOWNPURPOSES
Posted by MarioIncandenza on Wed, 23 Feb 2011 00:41:31 GMT
View Forum Message <> Reply to Message

DEFSYSV was new to me: it's what I ended up using for this job. But
the dist_tools look really promising, and I'll have to look harder at
them.

Very happy with this solution. Many thanks Mike and Jeremy!

--Edward H.

Subject: Re: subverting IDL builtin variables !FORMYOWNPURPOSES
Posted by SonicKenking on Wed, 23 Feb 2011 12:42:18 GMT
View Forum Message <> Reply to Message

> I ended up using common blocks for my logging framework, but system
> variables would work just as well.
>
> See MG_LOG and MGffLogger in the dist_tools for the way I did it:
>
>   http://docs.idldev.com/dist_tools/
>
> By the way, you don't have to subvert a pre-existing system variable,
> you can create your own with DEFSYSV.
>
> Mike
> --www.michaelgalloy.com
> Research Mathematician
> Tech-X Corporation

Hi Mike,

The MG_LOG is a really nice tool and inspiring. I did some
modifications to it and think it could be useful to others.

One limitation I felt for MG_LOG is that it accepts only a single string message to print. I'd like to have something similar to the IDL built-in PRINT, which accepts any number of arguments and also does the logging as MG_LOG.

This is one tricky thing I always struggled, i.e. you cannot get the full power of using variable length parameters without using "Execute". But Execute is something I try to avoid as much as possible, since it does not run on an IDL VM. Mike suggested before that this can be solved by using DLM or embeded C programs. But I have no idea how to do it. I would appreciate if anyone can come up with a universal wrapper for this and is willing to share with us. :)

Anyway, I ended up re-using PRINTF for the logging and sneak the modified logging program in the position of LUN. So it is something like:

```
PRINTF, myLun(), a, b, c, etc
```

Where myLun() is a similar routine to MG_LOG.

They are similar in that they are both a wrapper that manages an underlying logging object. They both control whether the message/ variables are actually printed out based on a logging level (1-5).

The differences are:
1. myLun() returns a lun for either the terminal (-1 or -2), or a file, or /dev/null for suppressing the message to be printed.
2. myLun() accepts a level parameter indicating the level of this message, e.g. myLun(5) for debugging level.
3. myLun() accepts a filename for logging in a file and once the file is set, it is persistent until the logging is explicitly redirected to another file/terminal.
4. The underlying objects can manage multiple output files. An new file will be opened if a new filename is passed to myLun().

Some of the limitations are:
1. It cannot keep the output to both terminal and files at the same without using "Journal", which is another thing I try to avoid.
2. Currently it only controls text printing. But I'd like to have similar mechanisms but work for plottings.

I'll post the code if anyone is interested.

Yang