Subject: Re: status of HDF5 *writing* support in IDL
Posted by Michael Wallace on Tue, 27 Apr 2004 23:24:51 GMT
View Forum Message <> Reply to Message

>> I'm still waiting for command line arguments and
>> nice-looking fonts in direct graphics. And those aren't
>> the only things missing. At least I can "fix" the command
>> line argument with a nice little Python wrapper.
>
> Would you post an example of your python wrapper?

No problem. It's not a high quality Python program, but it gets the job done. Enjoy!

```
#!/usr/bin/python
#
# NDL (Non-interactive Data Language)
#
# Sadly, IDL does not accept command line arguments. This has been the
# bane of my existence for the last couple days. However, there is
# hope and hope comes in the form of this Python script.
#
# This program accepts the name of an IDL procedure, executes the
# procedure and then exits. Additional command line arguments are fed
# as an array of strings to the IDL procedure via the ARGS keyword.
# Therefore, any IDL procedure which needs to accept command line
# arguments needs to define ARGS as a keyword. If there's a procedure
# named IDL_Proc which has the ARGS keyword defined, the following
# command will cause IDL_Proc to run and ARGS will be filled with the
# strings 'arg1' and 'arg2'.
#
# $ ndl IDL_Proc arg1 arg2
#
# When there are no extra command line arguments, the procedure is
# called without the ARGS keyword. This allows you to call code which
# you didn't write or can't edit for some reason. In fact, you can
# actually send an entire IDL command as the first parameter and watch
# it execute. For example,
#
# $ ndl "Print, 'Hello, World'"
#
# will print "Hello, World" to the terminal along with all the standard
# IDL licensing junk you see when starting IDL. If you don't want to
# see all of this, just pipe stderr to /dev/null or some log file. Just
# make sure to not do this if you're debugging something! ;-)
#
# $ ndl "Print, 'Hello, World'" 2> /dev/null
```

```
#
# In short, all this program does is open a pipe to IDL and send an IDL
# statement based on the arguments provided.  That's it.  It's just a
# way to make IDL look like it is accepting command line arguments.
# Nothing more.  Nothing less.
#
#
# March 2004
#


import os
import sys


# Usage statement
usage = "usage: %s idlprog [arg1 [arg2 [ ... ]]]" \
        %os.path.basename(sys.argv[0])


# Check that the name of the IDL program was provided
if len(sys.argv) < 2:
    print usage
else:
    # Open a pipe to an IDL process to write to
    fd = os.popen('idl', 'w')


    # If extra arguments are given, pass them via the ARGS keyword
    if len(sys.argv) < 3:
        fd.write(sys.argv[1])
    else:
        fd.write(sys.argv[1] + ', ARGS = ' + `sys.argv[2:]`)


    fd.close()
```