Introduction to the IRAF'S CL as a Programming Language

Viewgraphs and class notes can be found at

http:/ra.stsci.edu/CL_Class/cl.html

For more information, contact

help@stsci.edu

The Class: What We Hope To Accomplish

Main Goal

To write and execute a *procedure script*.

Topics Introduced

- Script creation
- Process I/O redirection
- Parameters
- Procedure scripts
- Control constructs: looping and decision making

To Begin: The Problem

Problem

Recalibrate archive FOS observations to use the new AIS flux calibration algorithm in `calfos`.

IRAF Commands

```
cl> hedit y2et0105t.d0h apr_corr PERFORM add+
cl> hedit y2et0105t.d0h ais_corr PERFORM add+
cl> hedit y2et0105t.d0h tim_corr PERFORM add+
cl> getreffile y2et0105t | upreffile verify-
cl> calfos y2et0105t y2et0105t_flx
```

Note

A *pipe* is used to redirect the standard output of the task `getreffile` to the standard input of the task `upreffile`.

Using CL Logging for Script Creation

Example:

```
cl> logfile="aiscor_1.cl"
cl> keeplog=yes
cl> hedit y2et0105t.d0h apr_corr PERFORM add+
cl> hedit y2et0105t.d0h ais_corr PERFORM add+
cl> hedit y2et0105t.d0h tim_corr PERFORM add+
cl> getreffile y2et0105t | upreffile verify-
cl> calfos y2et0105t y2et0105t_flx
cl> keeplog=no
cl> edit aiscor_1.cl
```

aiscor_1.cl

```
keeplog=yes
hedit y2et0105t.d0h apr_corr PERFORM add+
hedit y2et0105t.d0h ais_corr PERFORM add+
hedit y2et0105t.d0h tim_corr PERFORM add+
getreffile y2et0105t | upreffile verify-
calfos y2et0105t y2et0105t_flx
```

Executing aiscor_1.cl

```
cl> edit aiscor_1.cl
```

... Remove the `keeplog=yes` line...
... modify the image name...

```
cl> cl < aiscor_1.cl
```

Note

Process I/O redirection is used to redirect the standard input of the cl task to come from the script file.

Creating Scripts using `mkscript`

```
cl> mkscript aiscor_2.cl hedit
```

`...` `eparam` **is started to modify the** `hedit` **parameters ...**

```
cl> mkscript aiscor_2.cl getreffile append+
```

`...` `eparam` **for** `getreffile...`

```
cl> mkscript aiscor_2.cl upreffile append+
cl> mkscript aiscor_2.cl calfos append+
```

aiscor_2.cl (as directly produced by `mkscript`)

```
hedit ("y2et0105t.d0h", "apr_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

hedit ("y2et0105t.d0h", "ais_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

hedit ("y2et0105t.d0h", "tim_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

getreffile ("y2et0105t", long=no, cdbspar="")

upreffile ("", yes, template="*.d0h", verify=no)

calfos ("y2et0105t", "y2et0105t_flx")
```

aiscor_2.cl (after a bit of editing)

```
hedit ("y2et0105t.d0h", "apr_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

hedit ("y2et0105t.d0h", "ais_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

hedit ("y2et0105t.d0h", "tim_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

getreffile ("y2et0105t", long=no, cdbspar="") |
upreffile ("", yes, template="*.d0h", verify=no)

calfos ("y2et0105t", "y2et0105t_flx")
```

Executing aiscor_2.cl

The command:

```
cl> cl < aiscor_2.cl
```

Notes

- Useful when writing a script "from scratch"
- No need to remember the parameters for individual tasks; runs `eparam`
- Scripts are written in *program* mode (note the parenthesis)
- If anything special is required, i.e. I/O redirection, the script will have to be editted.
- Scripts are easier to convert to *procedure* scripts than scripts created through the use of logging.

CL’s Schizoid Nature

The CL operates in two modes: *command* mode and *program* mode.

COMMAND Mode

- Default mode.
- Minimizes syntax needed to enter basic commands.
- No access to CL parameters
- Example: `cl> dir *.d0h`

PROGRAM Mode

- Requires formal syntax to be used
- Allows access to CL parameters
- Evaluates expressions
- Example: `cl> dir ("*.d0h")`

aiscor_3.cl

```
hedit (s1, "apr_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

hedit (s1, "ais_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

hedit (s1, "tim_corr", "PERFORM", add=yes, delete=no,
       verify=no, show=yes, update=yes)

fparse (s1, verbose=no)
getreffile (fparse.root, long=no, cdbspar="") |
upreffile ("", yes, template="*.d0h", verify=no)

calfos (fparse.root, fparse.root//"_flx")
```

Executing aiscor_3.cl

```
cl> s1="y2et0105t.d0h"
cl> cl < aiscor_3.cl
cl> s1="y2et0106t.d0h"
cl> cl < aiscor_3.cl
```

Predefined User Parameters

Type	Parameters
Boolean	b1, b2, b3
Integer	i, j, k
Real	x, y, z
Character (string)	s1, s2, s3
List structured	list
struct	line

Expressions

Expressions are evaluated by the CL and return a value.
Use "`help mathfcns`" for list of built-in functions.

Examples

```
cl> =2 + 2
4
cl> s1="Hello, "
cl> s2 = s1//"world!"
cl> =s2
Hello, world!
cl> =sin(90*3.14159/180.)
0.99999999999912
cl> x=3.14159
cl> y=180
cl> z = x/y
cl> =sin(90*z)
0.99999999999912
```

Using Task Parameters

Relevant Code

```
fparse (s1, verbose=no)
getreffile (fparse.root, long=no, cdbspar="")
...
calfos (fparse.root, fparse.root//"_flx")
```

Notes

- A task parameter is no different than any other parameter.
- Tasks can modify their parameters.
- The task must be specified when accessing a task parameter.

First Procedure Script: aiscor_4.cl

```
procedure aiscor_4 (input)

file      input {prompt="D0H image to reset headers and calibrate"}

begin
          # Declarations
          file      pinput

          # Get interactive parameters
          pinput = input

          # Update the headers and recalibrate.
          hedit (pinput, "apr_corr", "PERFORM", add=yes)
          hedit (pinput, "ais_corr", "PERFORM", add=yes)
          hedit (pinput, "tim_corr", "PERFORM", add=yes)
          fparse (pinput, verbose=no)
          getreffile (fparse.root, long=no, cdbspar="") |
          upreffile ("", yes, template="*.d0h", verify=no)
          calfos (fparse.root, fparse.root//"_flx")
end
```

Executing aiscor_4.cl

```
cl> task aiscor_4 = aiscor_4.cl
cl> aiscor_4
D0H image to reset headers and calibrate: y2et0105t.d0h
cl> aiscor_4 y2et0106t.d0h
```

CL Procedure Script: `procedure`

Formal Definition

`procedure` *`taskname (query-par, ...)`*

`type query-par {properties}`

`type hidden-par {properties}`

`begin`

`body of procedure`

`end`

`task` Statement

Syntax

```
task [$]name = [dir$]name.cl
```

Notes

- Task name, rootname of the file, and the name of the procedure **must** be the same.
- If the procedure script has no task parameters, place the '$' in front of the task name.

Examples

```
cl> task $aiscor_3=aiscor_3.cl
cl> task write_paper=documents$write_paper.cl
```

aiscor_4.cl: Parameter Declarations

Relevant Code

```
procedure aiscor_4 (input)

file      input {prompt="D0H image to reset headers and calibrate"}

begin
          # Declarations
          file      pinput

          # Get interactive parameters
          pinput = input
```

Notes

- Query parameters appear in the argument list.
- Local parameters are declared after the `begin` statement.
- Query parameters will prompt for input everytime they are referenced.

aiscor_5.cl

```
procedure aiscor_5 (input)

file      input {prompt="List of observations to flux calibrate"}

bool      calibrate {yes,prompt="Run calfos to recalibrate?"}

struct    *in_list

begin
          # Declarations
          file      fname
          file      in_list_file
          file      pinput

          # Get interactive parameters
          pinput = input

          # Make temporary file names.
          in_list_file = mktemp ("tmp$Example")
```

aiscor_5.cl (continued)

```
          # Update the headers and recalibrate.
          files (pinput, sort=yes, > in_list_file)
          in_list = in_list_file
          while (fscan (in_list, fname) != EOF) {

              hedit (fname, "apr_corr", "PERFORM", add=yes)
              hedit (fname, "ais_corr", "PERFORM", add=yes)
              hedit (fname, "tim_corr", "PERFORM", add=yes)
              fparse (fname, verbose=no)
              getreffile (fparse.root, long=no, cdbspar="") |
                    upreffile ("", yes, template="*.d0h", verify=no)

              if (calibrate)
                    calfos (fparse.root, fparse.root//"_flx")
          }
          in_list=""

          # That's all folks.
          delete ("tmp$Example*", verify=no, >& "dev$null")
end
```

aiscor_5.cl: Hidden Parameter

Relevant Code

```
procedure aiscor_5 (input)
...
bool     calibrate {yes,prompt="Run calfos to recalibrate?"}
...
            if (calibrate)
                calfos (fparse.root, fparse.root//"_flx")
```

Notes

- Hidden parameters do not appear in the argument list.
- Default value specified for a parameter.
- Used directly in the script (as opposed to query parameters)

aiscor_5.cl: List Structure Parameter

Relevant Code

```
struct   *in_list
...
         files (pinput, sort=yes, > in_list_file)
         in_list = in_list_file
         while (fscan (in_list, fname) != EOF) {
...
         in_list=""
```

Notes

- Declared as a task parameter, not a local parameter.
- Initialized with the name of a file.
- After the loop is finished, the value is cleared, closing the file.

aiscor_5.cl: Control Constructs

Relevant Code

```
          while (fscan (in_list, fname) != EOF) {
...
              if (calibrate)
                   calfos (fparse.root, fparse.root//"_flx")
          }
```

Notes

- `while` statement used to loop through the list of files.
- `if` statement used to conditionally execute `calfos` task.
- `while` loop executes a `compound statement` as opposed to the `if` statement.
- Use of the instrinsic function `fscan` for CL input.

CL Control Structures: `while`

Formal Definition

```
while (expression)
    statement
```

Example

```
cl> list="alist.txt"
cl> while (fscan (list, s1) != EOF)
>>> print ("Next line from file is "//s1)
```

CL Control Structures: `if`

Formal Definition

```
if (expression)
    statement
[else
    statement]
```

Example

```
cl> if (i > 100)
>>> print ("i is too large")
>>> else
>>> print ("i is "//i)
```

Compound Statements

Allows one to define a *logical* statement.

Example

```
cl> {
>>> dir
>>> print ("Hello")
>>> display dev$pix 1
>>> }
cl> ^
```

... will do all three statements again ...

aiscor_3.cl: It Can Loop Too!

From the Command Line

```
cl> files *.d0h > files.list
cl> list = "files.list"
cl> while (fscan (list, s1) != EOF)
>>> cl < aiscor_3.cl
```

The Need for Information

IRAF On-line

The cl and its intrinsic functions are fairly well documented in IRAF's help database. Useful help pages include:

- commands
- language
- parameters
- "procedure" *Note: the quotes are necessary*
- task
- cursors
- mathfcns

To get a hard copy of any IRAF help file, remember

```
cl> help topic | lprint
```

Personal Creed (and USENIX motto)

"Use the Force, Read the Source"

How to View

For any task that is identified as a script, you can view the source with

```
cl> help taskname option=source
```

Some Scripts

- testhrs.paperprod
- testfos.yfluxcal
- onedspec.bplot
- ctools.fwplot

Exercises

Develop a degrees-to-radian procedure.

- Use `print` to print the result.
- Declare a hidden parameter and assign the result to it.

Find mean and standard deviation of a list of numbers

- Use `fscan` and a list structured parameter.

Find mean and standard deviation of a header keyword for a list of images

- Task `ttools.keypar` reads the value of a keyword and assigns it to the task parameter `keypar.value`
- Converting strings to reals: `x = real (s1)`