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

Topics

- Exercises
- Query Parameters
- Environmental Variables
- List Structured Parameters
- Input/Output
- Organizing scripts: packages
- Useful tasks
- Debugging
- Foreign tasks
- gcur

Exercise 1: deg2rad.cl

Source

```
procedure deg2rad (degrees)

real     degrees  {prompt="Angle (degrees)"}
real     radians  {prompt="Output: Angle (radians)"}

begin
         radians = 3.14159 / 180. * degrees
         print (radians)
end
```

Use

```
cl> task deg2rad=deg2rad.cl
cl> deg2rad 45
0.7853975
```

Exercise 2: mean_1.cl

```
# Determine mean/standard deviation of a list of numbers.

procedure mean_1(input)

file     input    {prompt="File containing list of numbers to calcu-
late"}

real     mean     {prompt="Output: mean of numbers"}
real     std      {prompt="Output: standard deviation"}

struct   *in

begin
         # Declarations
         int      n                  # Number of values accumulated.
         real     sum, sum2          # Sums.
         real     x                  # Value.
```

Exercise 2: mean_1.cl (continued)

```
# Get query parameters and initialize list.
in = input

# Collect statistics.
n = 0
sum = 0.
sum2 = 0.
while (fscan (in, x) != EOF) {
    sum = sum + x
    sum2 = sum2 + (x*x)
    n = n +1
}
```

Exercise 2: mean_1.cl (continued)

```
# Calculate statistics.
if (n == 0) {
    mean = INDEF
    std = INDEF
} else if (n == 1) {
    mean = sum
    std = INDEF
} else {
    mean = sum / n
    std = (sum2 - mean * sum) / (n - 1)
    if (std < 0)
         std = INDEF
    else
         std = sqrt (std)
}
in = ""
```

Exercise 2: Statistics (continued)

```
        # Print results
        print ("mean = "//mean//" std = "//std)
end
```

Exercise 2: mean_2.cl using a `switch`

Code Fragment

```
# Calculate statistics.
switch (n) {
case 0:
    {
            mean = INDEF
            std = INDEF
    }

case 1:
    {
            mean = sum
            std = INDEF
    }

default:
    {
            mean = sum / n
            std = (sum2 - mean * sum) / (n - 1)
            if (std < 0)
                std = INDEF
            else
                std = sqrt (std)
    }
```

Exercise 2: mean_3.cl using `tstat`

```
# Determine mean/standard deviation of a list of numbers.
procedure mean_3(input)

file        input       {prompt="File containing list of numbers to calculate"}

real        mean        {prompt="Output: mean of numbers"}
real        std         {prompt="Output: standard deviation"}

begin
            # Declarations
            file        pinput                  # Input file.

            # Get query parameters and initialize list.
            pinput = input

            # Calculate the statistics.
            tstat (pinput, "c1", outtable="", lowlim=INDEF, highlim=INDEF,
                   rows="-", n_tab="table", n_nam="column", n_nrows="nrows",
                   n_mean="mean", n_stddev="stddev", n_median="median",
                   n_min="min", n_max = "max")
            mean = tstat.mean
            std = tstat.stddev

            # Print results
            print ("mean = "//mean//" std = "//std)
end
```

Exercise 3: hdmean_1.cl

```
procedure hdmean_1 (input, keyword)

file       input     {prompt="List of images to get stats on"}
char       keyword   {prompt="Keyword to get stats on"}

real       mean      {prompt="Output: mean of numbers"}
real       std       {prompt="Output: standard deviation"}

struct     *flist

begin
           # Declarations
           file      flist_file, fname, pinput
           int       n
           char      pkeyword
           real      sum, sum2

           # Get interactive parameters
           pinput = input
           pkeyword = keyword
```

Exercise 3: hdmean_1.cl (continued)

```
# Get file list.
flist_file = mktemp ("tmp$HDMean")
files (pinput, > flist_file)

# Collect statistics
sum = 0.
sum2 = 0.
n = 0
flist = flist_file
while (fscan (flist, fname) != EOF) {
    keypar (fname, pkeyword, silent=yes)
    if (keypar.found) {
          n = n + 1
          x = real (keypar.value)
          sum = sum + x
          sum2 = sum2 + x * x
    }
}
```

Exercise 3: hdmean_1.cl (continued)

```
        # Calculate statistics.
        if (n == 0) {
            mean = INDEF
            std = INDEF
        } else if (n == 1) {
            mean = sum
            std = INDEF
        } else {
            mean = sum / n
            std = (sum2 - mean * sum) / (n - 1)
            if (std < 0)
                  std = INDEF
            else
                  std = sqrt (std)
        }
        flist = ""

        # Print results
        print ("mean = "//mean//" std = "//std)

        delete ("tmp$HDMean*", verify=no, >& "dev$null")
end
```

Exercise 3: hdmean_2.cl using `tstat`

```
procedure hdmean_2 (input, keyword)

file      input     {prompt="List of images to get stats on"}
char      keyword   {prompt="Keyword to get stats on"}

real      mean      {prompt="Output: mean of numbers"}
real      std       {prompt="Output: standard deviation"}

begin
          # Declarations
          file      kvalues, pinput
          char      pkeyword

          # Get interactive parameters
          pinput = input
          pkeyword = keyword
```

Exercise 3: hdmean_2.cl (continued)

```
          # Select the keyword values into a table.
          kvalues = mktemp ("tmp$HDMean")
          hselect (pinput, pkeyword, yes, > kvalues)

          # Calculate statistics
          tstat (kvalues, "c1", outtable="", lowlim=INDEF, highlim=INDEF,
                 rows="-", n_tab="table", n_nam="column", n_nrows="nrows",
                 n_mean="mean", n_stddev="stddev", n_median="median",
                 n_min="min", n_max = "max")
          mean = tstat.mean
          std = tstat.stddev

          # Print results
          print ("mean = "//mean//" std = "//std)

          delete ("tmp$HDMean*", verify=no, >& "dev$null")
end
```

listit.cl: Query Parameters

```
procedure listit (input, more)

file      input {prompt="File to list"}
bool      more {yes, prompt=">>>Next page?"}
struct    *flist

begin
          struct    line
          int       n, nlines

          flist = input

          more = yes
          nlines = int (envget ("ttynlines")) - 1
          n = 1
          while (fscan (flist, line) != EOF) {
              if (n % nlines == 0)
                    if (!more)
                        break
              print (line)
              n = n + 1
          }
end
```

listit.cl: Example Run

```
cl> task listit=listit.cl
cl> listit
File to list: login.cl

... page full of lines ...

>>>Next page? (yes): <return>

... another page ...

>>>Next page? (yes): n
cl>
```

Environmental Variables: Not Parameters

Environmental variables define the `environment` in which the CL is running.

Environmental variables are used to define a logical file structure and to modify the behavior of the CL and tasks running under its control.

EV's vs. Parameters

- Value cannot be accessed directly; need to use built-in `envget()`
- Value cannot be modified directly; need to use built-in `set/reset`
- Only one type: string.
- No other parameter features: range checking, prompting, etc.

Commands

- `show`
- `set, reset`

Environmental Variables: File Names

Logical Pathnames

```
cl> show graphcap
dev$graphcap
cl> show dev
iraf$dev

cl> set myscratch=/scratch/tib1/eisenham/
cl> dir myscratch
cl> copy dev$pix myscratch$mypix.hhh
```

Environmental Variables: Control

Examples

```
cl> show imtype
imh
cl> set imtype=hhh
cl> show terminal
xgterm
```

Variables to know about

- imtype - Default image type
- clobber - Allow existing files to be overwritten
- editor - default editor
- min_lenuserarea - Image header buffer
- stdplot - Default graphics hardcopy printer
- stdimage - Default buffer size for image display
- showtype - Show whether a command is a task, package, or pset

Lists: List Structured Parameters

List structured parameters allow the CL to read single lines from a file, i.e. a list.

Example

```
cl> struct *alist
cl> alist="home$login.cl"
cl> =alist
# LOGIN.CL -- User login file for the IRAF command language.
cl> =alist

cl> =alist
# Identify login.cl version (checked in images.cl).
cl> alist="home$login.cl"
cl> =alist
# LOGIN.CL -- User login file for the IRAF command language.
```

Lists: `fscan`

Format

```
fscan (param, p1, p2, p3, ... pn)
```

Examples

```
cl> s1="Hello, world!"
cl> =fscan (s1, line)
1
cl> =line
Hello, world!
cl> =fscan (s1, s2, s3)
2
cl> =s1
Hello,
cl> =s2
world!
```

Lists: List Structured Parameters + `fscan`

Example

```
cl> for (i=1; i<11; i=i+1)
>>> print (i, i+10, i**2, >> "num.txt")
cl>
cl> list="num.txt"
cl> while (fscan (list, i, j, k) != EOF
>>> print (i, j, k)
cl> list=""
```

More on Input/Output: `scan`

Format

n = `scan (`*p1*`, `*p2*`, ..., `*pn*`)`

n = `nscan()`

Example

```
cl> time | scan (line)
cl> = line
Wed 11:35:54 06-Mar-96
cl> =nscan()
1
```

Note

`nscan()` also can be used in conjunction with `fscan`.

Yet More Input/Output: print, printf

Format

print (*string*, *string*,..., *[I/O redirection]*)

printf (*format*, *p1*, *p2*,..., [*I/O redirection*])

Examples

```
cl> print ("Hello, world!")
cl> print ("i equals ’"//i//"’")
i equals '1'
cl> print ("i equals ’",i,"’")
i equals '1 '

cl> printf ("i equals ’%d’\n", i)
i equals '1'
```

Note: Type ’`help ttools opt=sysdoc`’ for string formatting information.

Formatting Directives: Hours/Minutes/Seconds

Directives

```
%h - Convert to hh:mm:ss format

%H - Convert to hh:mm:ss with a division by 15
```

Examples

```
cl> x = 235.34728935
cl> printf ("x = %h\n", x)
x = 235:20:50.2
cl> printf ("x = %H\n", x)
x = 15:41:23.3
```

Organizing Scripts: Simple

Process

- Place all scripts in a common directory. Example: home$scripts
- Modify loginuser.cl (or login.cl) to add the task definitions.
- Restart the CL

Example: The scripts directory

```
cl> cd home$
cl> mkdir scripts
cl> cd scripts
... place desired scripts in the directory
cl> dir
deg2rad.cl  hdmean_2.cl  mean_1.cl
hdmean_1.cl listit.cl    mean_2.cl
```

Organizing Scripts: loginuser.cl modifications

Add the following lines to loginuser.cl (or login.cl):

```
set scripts=home$scripts/
task deg2rad=scripts$deg2rad.cl
task hdmean_1=scripts$hdmean_1.cl
task hdmean_2=scripts$hdmean_2.cl
task listit=scripts$listit.cl
task mean_1=scripts$mean_1.cl
task mean_2=scripts$mean_2.cl
```

Package Definition: scripts

Process

- Place all scripts in a common directory. Example: home$scripts
- Create a package script. Example: scripts.cl
- Modify loginuser.cl (or login.cl) to define the package (and preload).
- Restart the CL

scripts.cl

```
#{scripts package definition

package scripts

task deg2rad = scripts$deg2rad.cl
task hdmean_1 = scripts$hdmean_1.cl
task hdmean_2 = scripts$hdmean_2.cl
task listit   = scripts$listit.cl
task mean_1   = scripts$mean_1.cl
task mean_2   = scripts$mean_2.cl

clbye()
```

scripts: Setup in the loginuser.cl

Place the following in the loginuser.cl (or login.cl):

```
set scripts=home$scripts/
task scripts.pkg = scripts$scripts.cl
```

Use

```
cl> scripts
deg2rad       hdmean_2      mean_1
hdmean_1      listit        mean_2
cl> listit
File to list:
```

Script-friendly Tasks

Tasks

- fparse - File name handling
- keypar, parkey, hselect - Header keyword access
- keytab, tabkey, tprint, tdump, tinfo - Table access
- access, files, match, count - File manipulation
- deftask, defpac, defpar - Task/parameter existence
- gstatistics, tstat, imcalc, tcalc - Data statistics and manipulation
- listpix - Image data retrieval
- igi - Graph generation.

Packages

- system, lists, proto, softools
- ttools, tools, ctools

Debugging

Tools Available

- `print` - Best used for runtime/calculation problems
- Comment Character '#' - Best for syntax errors
- spp-mode available for emacs - Helps prevent errors

Error Message

- Error on line <last line of file>: syntax error
- INTERNAL ERROR on line <past EOF>: parser gagged

Hints

- Develop and execute in stages.
- Write procedure scripts with a "`debug`" parameter.
- When all else fails, comment out everything, then uncomment from the top, line by line.

Foreign Tasks

Format

```
task $name = "$foreign"

task $name = "$taskname [argument modifiers]"
```

Example

```
cl> task $vi = "$foreign"
cl> vi login.cl
```

Example with argument modifier

```
cl> task $vi = "$foreign"
cl> vi home$login.cl
login: Undefined variable
cl> task $vi = "$vi $(*)"
cl> vi home$login.cl
```

Graphics cursors: gcur, imcur

List structured parameters that, when referenced, bring up the graphics or image cursor for input.

Example

```
cl> files u*.c1h > wfpc.list
cl> list = "wfpc.list"
cl> while (fscan (list, s1) != EOF) {
>>> display (s1, 1)
>>> =imcur
>>> }
```

Use in a Script

See the task `yddintplot` for a complicated use of `gcur`.

To see source, do '`help yddintplot opt=source`'

More information: `help cursors`

Script Writing: Guidelines

- Do not make the script do too much: keep problems well-defined
- Provide reasonable defaults for *hidden* parameters
- USE COMMENTS: Everyone, including the original author, benefits
- “Retrieve” all *query* parameters ASAP
- Delete all temporary files created by the script
- Clear (close) all list structured parameters
- Avoid using externally-declared parameters
- Specify all parameters for tasks used in the script

Parameter Data Types

Type	Declaration	Explanation
Integer	int	Integer
Real/Double Precision	real	Double precision floating point.
Character string	char string	Character strings
Boolean	bool	Boolean, yes/no
File	file	File names
Multi-token string	struct	Lines
Graphics Cursor	gcur	Graphics cursor input
Image Cursor	imcur	Image cursor input
List Structure	**type*	Multi-valued/list parameters
Parameter Sets	pset	Parameter sets not associated with any single task.

Data Type: struct

A *struct* is a "multi-token string": a string that contains more than one word, or token. A struct is **not** a "list structured parameter".

Example

```
cl> char astring
cl> scan (astring)
Enter a string
1
cl> =astring
Enter

cl> struct astruct
cl> scan (astruct)
Enter a string
1
cl> =astruct
Enter a string
```

Parameter Modes

The available parameter modes are as follows:

- "a" automatic: parameter inherits whatever mode the cl is in.
- "q" query: the value of the parameter is explicitly asked for.
- "l" learn: parameter retains the value assigned to it.
- "h" hidden: value is not explicitly queried for.
- "m" menu: run eparam to retrieve values.

A parameter may be assigned more than one mode, i.e. "ql" for a parameter to query for its value and then retain that value.