SPIDER tutorial

Syntax, doc files and important notions

Magali COTTEVIEILLE - September 2009

Launch / terminate SPIDER

- SPIDER expects all files within a session to have the same filename extension. You specify it when you launch spider:

```
% spider spi/dat @script
```

- You can execute a series of tasks from commands stored in a file (cf. above), or interactively:

```
% spider dat
.OPERATION:
…
```

- Certain operations may have written data in the SPIDER results file: `results.dat.0`. A different results file will be created with each SPIDER run by incrementing the final digit(s) of the file name.
- End your interactive SPIDER session/ your script:

  `EN / END` (the results file is deleted)

- Comments in SPIDER scripts: preceded by a ";"

Doc files: write with SD

Information obtained during a SPIDER session can be saved to document files (called doc files). Data is written to a doc file with the SD operation, which writes one line at a time. The first number (here 1), is the key. This doc file can be read by other SPIDER operations; the key tells SPIDER which line to read/write.

```
.OPERATION: SD 1, [mic], [defocus], [max]
.DOCUMENT FILE: doc001
```

```
;spi/xlk   10-AUG-2009 AT 18:08:45 defgrps.xlk
   1 3   71.000       17946.       29675.
   2 3   77.000       17667.       31564.
   3 3   47.000       15456.       22486.
   4 3   53.000       15271.       24292.
   5 3   65.000       14397.       27892.
   6 3   59.000       14289.       26093.
   7 3   54.000       13865.       24505.
   8 3   60.000       12957.       26327.
   9 3   66.000       12735.       28134.
  10 3   84.000       11822.       33624.
```

File Names

- SPIDER can substitute for any portion of the file name at run time using a value contained in a symbolic variable or register variable. Any text entered as part of a filename which is enclosed between "{" and "}" or "[]" brackets is presumed to be part of a desired substitution request.
- To substitute a register variable value into a file name, use the sequence `{****[register_var]}` where the "*" string denotes the number of digits for the substitution and the "`[register_var]`" denotes the register variable whose numerical value is to be substituted in place of the astericks.
- There may be any number of substitution strings within a single filename, e.g. the following is a valid filename:
  `/usr/dir{*[dirnum]}/abcd{****[filenum]}`

Image Stacks

- Multiple images can be stored within a single SPIDER "stack file". A stacked image is referred to as: [ABC]@[###] where [ABC] is a sequence of alphanumeric characters and [###] is a sequence of digits. The digits after the '@' symbol represent the image number within a stack. Stacked images can be used anywhere a normal SPIDER file name would be used.
  An example of a filename that denotes image number 4056 inside stack file fil001 is: fil001@4056.

Doc files: read with UD

```
; UD N counts the number of lines in the doc file, puts
result in [n]
UD N [n]
randoc

DO [idx] = 1,[n]    ; Loop for each line in the doc file

UD [idx],[rotang]
randoc

RT
filt001
rot{***[idx]}
[rotang]
ENDDO

EN D
```

See also `UD IC / UD ICE`

Inline files:

Ex:

```
RT
test001
_1          ; ***** Replace the output filename with _1
60
```

If the images are large, and there are many operations, you can clutter your disk drive with numerous unnecessary files. SPIDER allows special variables to be substituted for filenames. These are called inline files, and are denoted by _N, where N is a digit from 1..99, preceded by one to three underscore symbols.

Variables

- Two types of variables are available in SPIDER.
  - Register (numerical) variables can contain any floating point value
  - Symbolic (String) variables are strings of characters.
- Both types of variables are denoted with '[]' brackets.
- You should not simultaneously use the same variable name for both a symbolic variable and a register variable within a single procedure
- Variable names can contain alphanumeric characters plus '_' and '-'
- Variable names should start with a alphabetic letter not a digit
- Other special characters including a blank may work but are not supported and may cease to work in the future
- Names are case-sensitive.

Register (numerical) variables

- Register variables now have user specifed names. These variables are denoted with '[]' brackets
- Old syntax, still valid: numerical register notations (e.g. X11)
- Ex: [size]=33 ; Create register variable

Symbolic (String) Variables

- Symbolic (String) Variables are denoted with '[]' brackets e.g. [filename].
- Syntax: example 1:

```
[filename] = 'PIC001'      ; Set a string variable
 This is equivalent to the obsolete sequence:
FR L          ; Set a string variable
[filename]PIC001
```

- Example 2:

```
[filename] = 'PIC{***[filenum]}' ; Set a string variable
```

- Example 3:

```
GLOBAL [dir] = 'img'       ; Set a global string variable
GLOBAL [global_file]= '[dir]/IMG055'     ; Set a global
string variable
```

- See in index of operation: operation `VAR`

Control Structures: DO Loops

- DO loops enable operations to be repeated. The DO loop syntax is:

```
DO  <loop register> = <start_value>, <end_value>
   < Any SPIDER operations can be inside the DO loop >
ENDDO
```

- An older DO loop syntax that you may still see is is:

```
DO <Label> <loop register variable > = <start_value>,
<end_value>
   < Any SPIDER operations can be inside the DO loop >
<Label>
```

The IF Statement

- The IF operation is a control structure that provides conditional execution. The IF syntax is:

```
IF (TEST_CONDITION) THEN
   <any SPIDER operations>
ENDIF
```

- where the test condition tests the value of an expression (see example below). The SPIDER operations inside the IF statement are only executed if the condition is true, otherwise processing just moves to the next operation after the IF.
- Application: test the existence of a file; if it exists, do operations:

```
IQ FI [exists] ; If the following file exists, set [exists]=1, else [exists]=0
randoc
IF ([exists].EQ.1) THEN
DE
randoc
ENDIF
```

Most common error messages

- `*** FILE NOT FOUND: test999.dat`
  Check your inputs!!!
- Number of stars in the symbolic (string) variables don't match the number of digits in the name of your images.
- You forgot ( ) to define values in loops
- Don't forget to open your results file. If you have plenty, and you don't remember which one is the last one: `ls -lhtr res*`

Procedure files

- Procedure files are subroutines that can be called from other procedure files. They can have variable arguments that change with every call to the procedure.
- If a procedure is called from interactive session, when a 'RE' operation is encountered inside a procedure, control will return to the terminal. If a procedure is called from another procedure file, control will be passed to the operation following the procedure call.
- See http://www.wadsworth.org/spider_doc/spider/docs/quickstart.html#PROCS for more

Register variable argument transfer to procedures

- To pass initial arguments from the procedure caller to the called procedure, the user puts the arguments (if any) needed by the procedure behind the procedure name, enclosed in parenthesis, in the same order as they appear in the called procedure file. Procedures may pass up to 12 register variable arguments if these are matched by a defining argument sequence in the procedure called. The defining argument sequence must appear as the first line of the called procedure, and must be of the form:

  `([<N1>],[<N2>],[<N3>],...)`

  Where: `[<N1>],[<N2>],[<N3>]`,... are register variables appearing in the procedure. NOTE: Prior to 2004 '[]' brackets were used instead of '()'. The calling sequence must have the same number of registers, and must be of the form:

  `@<PROCNAME>>([<M1>],[<M2>],[<M3>],...).`

- Other possibility: operation RR

  Example: `lfc pick.spi` calling `pickparticle.spi`:

```
; FIND IF A SELECTION DOC FILE USED.
RR X41
? DO YOU WANT TO USE A SELECTION FILE (NO = 0, YES = 1) ?
```

Symbolic (string) variable transfer to procedures

There are 3 methods to pass symbolic variables to a called procedure.

- **1)** Replace the desired parameters inside the called procedure with a solicitation prompt and variable name for the parameters. Place the variable names behind the procedure calling statement in the exact same order as they will be queried in the called procedure. Inside the called procedure indicate that the variables should be obtained from the caller by placing a solicitation prompt enclosed in question marks followed by a symbolic variable name e.g., `?ROTATION ANGLE?[angle]` in place of the variable that the user wishes to read.
- Example: if the procedure TEST_MASTER contains the following:

```
@TEST_VAR       ; Procedure
PIC002    ; 1st solicited parameter (input filename)
30              ; 2nd solicited parameter (angle)
EN
```

- and procedure: TEST_VAR contains the following:

```
RT                ; Rotate images operation
?INPUT FILE?     ; Solicit input filename
OUT007            ; Output filename
?ROTATION ANGLE?[angle]; Solicit rotation angle and assign to symbol: [angle]
RE
```

- Then
  `OPERATION: @TEST_MASTER` will read the image PIC002, rotate it by 30 degrees and store it in: OUT007.

Symbolic (string) variable transfer to procedures

There are 3 methods to pass symbolic variables to a called procedure.

- **2)** Use the FR operation inside the called procedure to create a local symbolic variable by reading its value from a calling procedure. Ex:
  lfc_pick.spi calling pickparticle.spi:

```
FR
?MICROGRAPH FILE (INPUT) NAME ?<1>
```

- **3)** Global variable assignment from the master procedure:
  - Old syntax:
    FR G - Read (sets) a global string (symbolic) variable:
    Ex:

```
FR G
[raw_file]img{***[iter]}
```

  - New syntax:

```
GLOBAL [raw_file] = 'img{***[iter]}'   ; Set a global string variable
```