# Lesson 3-1: SAC Macros

SAC has a facility like Unix shell scripts or Windows .bat files.

Repeated or commonly used sequences of commands can be stored in a file and invoked by file name.

Macro files can have optional parameters and can use variables internally like programs.

SAC command to invoke a macro is `MACRO` or `M`

```
SAC> M TTSAC  ;* Invoke macro in file named TTSAC
```

# Macro files

- Text files, not .rtf or .doc
- No particular file suffix required (no .txt or .m), so optional use as to taste
- Edit file with usual text editors (TextEdit, BBEditLite, `vi`, `nedit`, `pico`)
- Lines contain SAC commands or small set of macro commands only recognized in macro files
- Macro commands begin with “$” to distinguish

# Example

- `Hello` macro
  - enter following text in a file called `Hello` in your working directory:

```
MESSAGE “The macro is starting …”
$RUN cat -
Hello, World!
$ENDRUN
MESSAGE “… and now it is finished.”
```

  - Invoke macro using MACRO command

```
SAC> M Hello
```

  - SAC commands used (`MESSAGE` writes a message to the user); `MACRO` command used (`$RUN` runs the program “`cat`” with input up to `$ENDRUN`)

# My result ...

```
SAC> sc vi Hello            ;* Use vi command to create file

SAC> m Hello                ;* Invoke

 The macro is starting ...
Hello, World!
 ... and now it is finished.

SAC>
```

# Seeing macros operate

- Use SAC command `ECHO ON` / `ECHO OFF`
- Each line is echoed as it is retrieved from the macro file
- Each line also echoed after macro variables are substituted into the text
- Useful for debugging / developing macros

# My result with ECHO ON ...

```
SAC> echo on                    ;* Turn on macro line echoing

SAC> m Hello                    ;* Invoke
 m Hello
 MESSAGE "The macro is starting ..."
 The macro is starting ...
 $RUN cat -
 Hello, World!
 $ENDRUN
Hello, World!
 MESSAGE "... and now it is finished."
 ... and now it is finished.
SAC>
```

- Lines in red are echoed before processing
- Even command lines are repeated
- Lines with macro variable substitutions will be prefixed with ==> (none seen here)

# Macro searches

- Macro files searched for in particular directories, and in particular order:
  - current directory (where SAC run from)
  - directories specifically indicated to SAC
  - default, built-in macro directory

- Special macro search directories designated with `SETMACRO` command:

```
SAC> SETMACRO /U/kit/sacmacro  ;* Set search path

SAC> SETMACRO MORE /tmp        ;* Add another path element

SAC> * Macro search path now is current directory,
SAC> *   /U/kit/sacmacro, /tmp, and default built-in macros.
```

- Commonly used in SAC startup files

# Conditional commands in macros

- `IF` / `ENDIF` and `IF` / `ELSE` / `ENDIF`
  - Basic conditional commands

- `IF` / `ELSEIF` / `ELSEIF` / `ELSE` / `ENDIF`
  - Select one case from many
  - Similar to C `switch`, shell `case`, Fortran `if`

- `DO` / `ENDDO` and `WHILE` / `ENDDO`
  - Basic looping constructs

# Variables in macros

- Three types of variables:
  - Blackboard variables – reference syntax `%X%`
  - Macro variables – reference syntax `$X$`
  - File variables – reference syntax `&n,X&`

- Blackboard variable properties
  - Global in scope (can be referred to on command line and in macros)
  - User can set and retrieve values

- Macro variables
  - Local in scope (only exist inside a macro)
  - User can retrieve values but not settable

# Variables in macros

- File variables
  - Global in scope but local to each SAC trace in memory
  - Syntax `&n,x&` refers to value X in file n (n can be a file number or file name)
  - User can retrieve values only
  - Names (here, `x`) restricted to file header variable names

# Setting variables in macros

- Blackboard most commonly used
- SETBB command sets value, %…% uses value

```
SAC> SETBB x “hello” ;* Set x to character string “hello”

SAC> MESSAGE “I said %x% to you!” ;* value of x substituted

 I said hello to you!
```

- Numeric example

```
SAC> SETBB sum 0 ;* Set sum to value zero
SAC> SETBB sum (%sum% + 1)    ;* increment sum
SAC> SETBB sum (%sum% + 1)

SAC> MESSAGE “sum is %sum%”   ;* value of sum substituted

 sum is 2.000000
```

# Unsetting variables in macros (and elsewhere)

- Blackboard variables global in scope, so those used in macros hang around forever
- UNSETBB command expunges definition

```
SAC> SETBB x “hello” ;* Set x to character string “hello”
SAC> MESSAGE “I said %x% to you!” ;* value of x substituted
 I said hello to you!
SAC> UNSETBB x                    ;* unset
SAC> MESSAGE “I said %x% to you!” ;* x forgotten
 ERROR 1201: Could not find VARS variable blackboard X
SAC>* No longer insistent, it seems
```

# Expressions

- Always appear inside parentheses (…)
- Can appear anywhere
- Can be nested
- Can be numeric or character valued
- Numeric expressions
  - ( x + y ), ( x – y ), ( x * y ), ( x / y ), (x ** y )
- Function expressions (names are case insensitive; here x is an arbitrary number)
  - (SQRT x), (EXP x), (ALOG x), (POWER x), (ALOG10 x)
  - (SINE x), (COSINE x), (TANGENT x), (ARCSINE x), etc.

# More expressions

- Miscellaneous functions (a, b, c, x numbers)
  - (MINIMUM a b c …), (MAXIMUM a b c …)
  - (INTEGER x), (ABSOLUTE x)
  - (PI)
  - (STATUS NFILES) number of files in SAC memory
- Time series query functions
  - (GETTIME x) returns time in each file where value of x is encountered
  - (GETTIME MAX) returns time where maximum value is found
  - (GETVAL y) returns value in each file at time y
  - (GETVAL FILE n y) same, but only in file n

# Still more expressions

- Character manipulation expressions
  - (SUBSTRING m n s) return characters m to n in character string s (1 is start, END is end)
  - (CHANGE x y s) change string x to y in string s
  - (DELETE x s) deletes string x in string s
  - (BEFORE x s) returns all of string s up to x
  - (REPLY x) type x on terminal and return reply
  - (ITEM n s) return blank-delimited item n from string s
- see HELP EXPRESSIONS for full list

# Still more expressions

- Header selection expressions
  - (`HDRVAL` h op x op y ...) queries header variable h of all files in memory and compares using op with value x and then op with y; all successful values returned.
    - op can be `LT LE EQ GE GT NE AZ PM`
    - `AZ` x `PM` y means azimuth is x±y; circular comparison done
  - (`HDRNUM` h op x op y ...) queries header like `HDRVAL`; all successful file numbers returned.

# Still more expressions

- HDRVAL / HDRNUM examples
  - 4 files in memory:

```
SAC> lh evdp

  FILE: /tmp/ex-1
  ---------------
    evdp = 10.0

  FILE: /tmp/ex-2
  ---------------
    evdp = 50.0

  FILE: /tmp/ex-3
  ---------------
    evdp = 250.0

  FILE: /tmp/ex-4
  ---------------
    evdp = 640.0
```

```
SAC> message "(hdrval evdp ge 200)"
 250. 640.
SAC> message "(hdrnum evdp ge 200)"
 3 4
SAC> * files 3 and 4 in list have evdp > 200
SAC> message "(hdrnum evdp gt 35 le 200)"
 2
SAC> * only file 2 has evdp > 35 and <= 200
```

# Compounded expressions

- SAC processes expressions in three stages:
  - expansion of blackboard variables %..% and macro variables $..$
  - expansion of file header variables
  - evaluation of functions
- Blackboard or macro variables can be inside file header variables

```
SAC> setbb x 1                                    ;* set x to 1
SAC> message "File %x% evdp is &%x%,evdp&"  ;* %x% is inside &..&
 File 1 evdp is 10.0
SAC>
```

- Expressions nestable

```
SAC> setbb x 1                                    ;* set x to 1
SAC> message "sqrt (%x% + %x%) is (sqrt (%x% + %x%))"
 sqrt 2.000000 is 1.414214
SAC>
```

# Escape character @

- Prevent interpretation of `(xxx)` as expression or function
- Prevent interpretation of `%xx%`, `$xx$`, `&xx&` as macro variables
- Prefix any dangerous character with @ to inhibit interpretation:

```
SAC> message "This is a 7@% solution NOT a bb var"

 This is a 7% solution NOT a bb var

SAC>  message "This is a 7% solution NOT a bb var"

 ERROR 1201: Could not find VARS variable blackboard

SAC>
```

# Conditional commands: conditions

- Relational conditions used in `IF` / `ELSEIF` / `WHILE` commands
- Syntax is e1 op e2
  - e1, e2 are numerical or character expressions
  - op is relational operator:
    - `EQ` (equal), `NE` (not equal)
    - `LT` (less than), `LE` (less than or equal)
    - `GT` (greater than), `GE` (greater than or equal)
  - (inspired by Fortran syntax)

# Conditional commands: conditions

- Examples:

```
if “%x%” EQ ‘debug’    ;* quotes in case string contains blanks
   setbb debug 1
endif

* Below, reply is prefixed with _ to recognize no response
*   -- a blank line typed for the reply
setbb ans “_(reply ‘Enter yes or no:’)”   ;* prompt and get answer
if “%ans%” eq “_yes”
   message “response is yes”
elseif “%ans%” eq “_no”
   message “response is no”
elseif “%ans%” eq “_”
   message “nothing typed”
else
   message “invalid response: %ans%”
endif
```

# Macro suspension and resume

- Set of macro commands to control macro operations:
  - `$KILL` – terminate macro
  - `$TERMINAL` – suspend macro for a while and take typed commands from keyboard
  - `$RESUME` – resume running of suspended macro

# Interacting with your OS

- SYSTEMCOMMAND is the key command
  - SAC assembles a command and optionally retains output from the command
  - Example:  Using UNIX date command

```
SAC> systemcommand cat /tmp/demosc
message "Time now is:"
systemcommand date              ;* Invoke date command

SAC> m /tmp/demosc
 Time now is:
Thu Jun 17 09:43:27 BST 2010
SAC>
```

- SC is abbreviation for SYSTEMCOMMAND
  - will see often in coming material

# Conditional commands: looping

- Example: `WHILE` condition / `ENDDO`

```
SAC> sc cat /tmp/xample
setbb x 1
while %x% LT 4
   message "x is now %x%"
   setbb x (%x% + 1)
enddo

SAC> m /tmp/xample
 x is now 1
 x is now 2.000000
 x is now 3.000000
SAC>* Note how arithmetic adds trailing zeroes
```

# Conditional command and macro suspension example

```
message "Fee fie foe fum" "I smell the blood of an Englishman"
message "What should I do now?"

* Ask for response and validate
setbb ok no                                ;* Sets value of bb var ok
while %ok% EQ no                           ;* Tests value of bb var ok
   setbb ans "_(reply 'respond kill or pause:')" ;* Prompt keyboard
   if "%ans%" EQ _kill
      setbb ok yes
   elseif "%ans%" EQ _pause
      setbb ok yes
   else
      message "Invalid response"
   endif
enddo

* Act on response
if "%ans%" EQ _kill
   message "OK, quitting ..."          ;* Macro will terminate
   $KILL
endif
if "%ans%" EQ _pause
   message "Type @$RESUME to resume"   ;* Macro will be suspended
   $TERMINAL
   message "Resuming macro ..."        ;* Resumed at this point
endif
message "If he is alive or if he is dead"
message "I'll crush his bones to make my bread"
```

# Conditional commands: looping

- Example: `WHILE READ` bb x y … z / `ENDDO`

```
SAC> sc cat /tmp/xample
setbb inp "one two three and the rest"

while read inp a b c d   ;* bb variable read is %inp%, $a$, $b$
   message "a is $a$"    ;* note macro variable, not bb variable
   message "b is $b$"
   message "c is $c$"
   message "and d is '$d$'"
enddo

SAC> m /tmp/xample
 a is one
 b is two
 c is three
 and d is 'and the rest'
SAC>
```

- Typical use is to put Unix command/program output into bb variable and process it line-by-line
  - see `SYSTEMCOMMAND` for details

# Conditional commands: looping

- DO / ENDDO
  - Gives ability to loop over finite set of values
  - Can loop over implicit or explicit list of values
    - Implicit (numeric):
      - DO v FROM x TO y / DO v FROM x TO y BY z
        - » v is macro variable, x, y, z are numeric expressions
        - » v successively takes on implied sequence of values
    - Explicit (textual):
      - DO v LIST a b c …
        - » macro variable v takes value a, then b, then c, …
      - DO v WILD a b … / DO v WILD DIR d a b …
        - » macro variable v takes on names of files that match wild card expression a, then b, …
- BREAK
  - Leaves DO or WHILE loop early

# Conditional commands: looping

- Example: `DO .. FROM / ENDDO`

```
SAC> sc cat /tmp/dofrom
setbb fib 1.0 prefib 0
do i from 0 to 10              ;* Loop macro variable value is $i$
   message "Fibonacci number $i$ is (BEFORE . %fib%)"
   setbb newfib (%fib% + %prefib%)          ;* Next Fibonacci num
   setbb prefib %fib% ; setbb fib %newfib% ;* Remember previous two
enddo

SAC> m /tmp/dofrom
 Fibonacci number 0 is 1
 Fibonacci number 1 is 1
 Fibonacci number 2 is 2
 Fibonacci number 3 is 3
 Fibonacci number 4 is 5
 Fibonacci number 5 is 8
 Fibonacci number 6 is 13
 Fibonacci number 7 is 21
 Fibonacci number 8 is 34
 Fibonacci number 9 is 55
 Fibonacci number 10 is 89
SAC>* Tip:  using (BEFORE . x) function strips trailing zeroes
```

# Conditional commands: looping

- Example: `DO .. FROM / ENDDO` using `BREAK`

```
SAC> sc cat /tmp/dobreak
setbb num "(reply 'Enter number:')"
do pow from 1 to 32       ;* Try powers of two
   setbb v (2 ** $pow$)
   if %v% GE %num%        ;* Test present one
      break               ;* Equals or exceeds number
   endif
enddo
message "Power of 2 larger than %num% is 2**$pow$ or %v%"
SAC>

SAC>  m /tmp/dobreak
Enter number:2096
 Power of 2 larger than 2096 is 2**12 or 4096.000
SAC>
```

# Conditional commands: looping

- Example: `DO .. LIST / ENDDO`

```
SAC> sc cat /tmp/dolist
do hdr list stla stlo evla evlo evdp scale ;* Header fields to check
   if UNDEFINED EQ "&1,$hdr$&"             ;* Check if set
      message "$hdr$ not set in file header"
   endif
enddo
SAC>

SAC> funcgen seismogram                    ;* Built-in seismic data
SAC> m /tmp/dolist
 scale not set in file header
SAC>* SCALE is rarely used in file headers
```

# Parameters to macros

- Macros can take parameters, like shell scripts
- Parameters can be positional or keyword
  - positional example:  1st parameter is name, 2nd is frequency

```
SAC> m example KEVO.BHZ 2.5
```

  - keyword example: keyword `FILE` identifies name, `FREQ` identifies frequency

```
SAC> m example file KEVO.BHZ freq 2.5
SAC> m example freq 2.5 file KEVO.BHZ  ;* keyword order independent
```

# Parameters to macros

- Parameters may be required or optional
  - Required parameters, if not provided, are prompted for when used
  - Optional parameters have defaults given in the macro
- Positional parameters simplest
  - Values obtained in macro using $1$, $2$, ...

```
SAC> sc cat /tmp/mppos                          ;* Macro is file /tmp/mppos
* Macro user provides two parameters:  SAC file and frequency
message "File name is $1$; frequency is $2$"

read $1$                                        ;* Read SAC file
rmean; rtrend terse; taper w 0.05               ;* Prepare for filtering
lowpass corner $2$ npoles 2 passes 2            ;* Filter corner frequency

SAC> m /tmp/mppos /tmp/ex-1 2                   ;* Macro used here
 File name is /tmp/ex-1; frequency is 2
SAC>
```

# Parameters to macros

- Prompt if required parameter missing:

```
SAC> m /tmp/mppos /tmp/ex-1              ;* Second parameter missing
2?  5
 File name is /tmp/ex-1; frequency is 5

SAC>* Prompt “2?” requests value for second parameter:  5 given
```

- Optional parameters indicated by providing default value using `$DEFAULT` command:

```
* Macro user provides two parameters:  file and frequency
$default 2 5                                ;* 2nd par. default 5
message "File name is $1$; frequency is $2$"

read $1$
rmean; rtrend terse; taper w 0.05
lowpass corner $2$ npoles 2 passes 2
```

# Parameters to macros

- Examples using defaulting:

```
* Macro user provides two parameters:  file and frequency
$default 2 5                                  ;* 2nd par. default 5
message "File name is $1$; frequency is $2$"

read $1$
rmean; rtrend terse; taper w 0.05
lowpass corner $2$ npoles 2 passes 2
```

```
SAC> m /tmp/mppos /tmp/ex-1                   ;* Second par omitted
 File name is /tmp/ex-1; frequency is 5

SAC> m /tmp/mppos                             ;* Both pars omitted
1?  /tmp/ex-1
 File name is /tmp/ex-1; frequency is 5

SAC>* Prompt for first parameter which lacks a default

SAC> m /tmp/mppos /tmp/ex-1 1
 File name is /tmp/ex-1; frequency is 1
SAC>* Explicit second parameter value overrides default
```

# Parameters to macros

- Keyword parameters are position independent
  - Text following keyword up to next keyword becomes value for parameter
  - Keyword declared with `$KEYS` command in macro

```
SAC> sc cat /tmp/mpkey                      ;* Macro file is /tmp/mpkey
* Macro user provides file xxxx and freq yyyy
$keys file freq
message "File name is $file$; frequency is $freq$"

read $file$
rmean; rtrend terse; taper w 0.05
lowpass corner $freq$ npoles 2 passes 2

SAC> m /tmp/mpkey file /tmp/ex-1 freq 5
 File name is /tmp/ex-1; frequency is 5

SAC>
```

# Parameters to macros

- Missing keyword parameter prompts are more meaningful

```
SAC> m /tmp/mpkey file /tmp/ex-1          ;* Only one keyword given
freq?  2
 File name is /tmp/ex-1; frequency is 2

SAC>
```

- Keyword parameter defaults given with $DEFAULT as well

```
* Macro user provides file xxxx and freq yyyy
$keys file freq
$default freq 5                          ;* default frequency is 5
message "File name is $file$; frequency is $freq$"

read $file$
rmean; rtrend terse; taper w 0.05
lowpass corner $freq$ npoles 2 passes 2
```

# Macro vs blackboard variables

- Blackboard variables `%xxx%` set/changed and used by user
  - scope is global
- Macro variables `$xxx$` set/changed by system, used by user
  - scope local to macro
  - set when `MACRO` command seen based on parameters
  - set by `DO` / `WHILE` commands

# Advanced interaction with your OS

- SYSTEMCOMMAND / SC is key command
- Can use system commands and capture their output inside of SAC

```
SAC> sc to out date   ;* send date program output to bb var out
```

  - Example:  Capture UNIX date command for nicer formatting

```
SAC> sc cat /tmp/demosc
sc to out date                          ;* date output to bb var out
message "Time now is: (ITEM 4 %out%)" ;* select 4th field in %out%

SAC>  m /tmp/demosc
 Time now is: 09:51:29
SAC>
```

# Advanced interaction with your OS

- Process multi-line `SC` output using `WHILE READ` loops
- Example: Handling `ls` command output

```
SAC> sc ls -l /tmp/do*              ;* output from ls command
-rw-------  1 geo-g4  wheel  210 Jun 17 09:18 /tmp/dobreak
-rw-------  1 geo-g4  wheel  179 Jun 17 08:55 /tmp/dofrom
-rw-------  1 geo-g4  wheel  309 Jun 17 10:06 /tmp/doscls

SAC> sc cat /tmp/doscls
sc to lsout ls -l /tmp/do*          ;* output saved in bb var lsout

while read lsout fperm flink fowner fgroup fsize fmm fdd ftt fname
   * Field 1 of output [permissions] assigned to fperm
   * Field 2 of output [links] assigned to flink
   * Field 3 of output [user name] assigned to fuser, etc.
   message "File $fname$ size $fsize$"
enddo

SAC>  m /tmp/doscls
 File /tmp/dobreak size 210
 File /tmp/dofrom size 179
 File /tmp/doscls size 309
```

# Interacting with your OS through command input

- OS commands invoked in macros can also take an input stream embedded in the macro
- `$RUN` / `$ENDRUN` macro commands bracket input lines
- Macro variable substitution done on input – makes it easy to pass information to programs
- Example:

```
SAC> sc cat /tmp/dorun
setbb fn /tmp/seismo
do sfx list BHE BHN BHZ ;* cat input file
   $RUN cat -
File is %fn%.$sfx$
   $ENDRUN
enddo

SAC> m /tmp/dorun        ;* runs cat 3 times
File is /tmp/seismo.BHE
File is /tmp/seismo.BHN
File is /tmp/seismo.BHZ
SAC>
```

# Interacting with your OS through command input, fine-tuned

- Copy `$RUN` / `$ENDRUN` input to temporary file
- Invoke Unix command with input from temporary file
- Uses Unix shell features to build temporary file names:

```
# echo /tmp/temp$$      # shell changes $$ to unique number
/tmp/temp372
#
```

  - Using SC, have to escape $$ with @ to prevent macro variable interpretation

```
SAC> sc echo /tmp/temp@$@$
/tmp/temp41534
SAC>
```

# Example (macro)

```
sc to scr echo /tmp/temp@$@$.sh          ;* Temp script file name %scr%

* Copy text to create a temporary shell script to test file existence
*   Note need to use escape characters for shell script syntax.
$run cat - > %scr%
# This shell script reads a file name and checks whether it exists
#   It echos the file name and adds "yes" if it exists or "no"
while read fn ; do
   test -f \@$fn @&@& echo \@$fn yes || echo \@$fn no
done
$endrun

sc to finp echo /tmp/temp@$@$.in         ;* Temp input file name %finp%
setbb pfx /tmp/event
do sfx list BHE BHN BHZ
   sc echo %pfx%.$sfx$ >> %finp%         ;* Add file name to input file
enddo

sc to inp sh %scr% < %finp%              ;* Run script on input file

while read inp fn exist                  ;* Process output from script
   message "File $fn$ exists? $exist$"   ;* Report file existence
enddo

sc rm %scr% %finp%                       ;* Remove temporary files
```

# Example (output)

```
SAC> m /tmp/ftester
 File /tmp/event.BHE exists? no
 File /tmp/event.BHN exists? no
 File /tmp/event.BHZ exists? no
SAC>
```

- Could use similar script to verify that three component data exists for each event to be studied.