GrADS scripting language reference card version 1.7

(GrADS Version 1.7 beta 7) compiled by *Karin Meier-Fleischer*, DKRZ (beratung@dkrz.de

General Information

The GrADS scripting language, used via the GrADS run command, provides a similar capability to the exec command, except that a script may have variables, flow control and access GrADS command output. Scripts may be written to perform a variety of functions, such as allowing a user to point and click on the screen to select something, to animate and desired quantities, to annotate plots with information obtained from GrADS query commands.

Important: GrADS needs a carriage return after the last command line in the script file, otherwise GrADS won't execute this command line.

Variables

Script language variable names are 1 to 8 characters, beginning with an alphabetic character and containing letters or numbers only. The name is case sensitive. The contents of a script variable is always a character string! For some operations, the character string will be interpreted as a number.

Predefined variables
lat
lon
lev
result
rec

String variables or **string constants** are enclosed either with single or double quotes.

name = 'Peter Pan'
or **name = "Peter Pan"**

Compound variables can be used to construct arrays in scripts. A compound variable has a variable name with segments seperated by periods.

varname.i.j

Example:

```
i = 10
j = 3
varname.i.j = 343
or varname.10.3 = 343
```

Note: The compound variable name MAY NOT be longer than 16 characters either BEFORE or AFTER substitution. GrADS scripting language is not particular efficient in handling large numbers of variables. Thus compound variables should not be used to create large arrays!

Global variables start with an underscore (_) and will keep its value throughout an entire script file using (also in functions).

_varname

Example: `_var1 = 1024`

Note: The global variables cannot be used in function headers
'function myfunc (_var1)' would be invalid!
It wouldn't make sense, cause it's a global variable!!!

Assignment

The format to assign a record is: **variable = expression**

The expression is evaluated, and the result is assigned to be the value of the indicated variable.

Logical values

Logical values are

TRUE	**1**
FALSE	**0**

Operators

The following operators are implemented:

|	logical OR	&	logical AND
!	unary NOT	-	unary minus
%	concatenation	=	equaltity
!=	not equal	>	greater than
>=	greater than or equal	<	less than
<=	less than or equal	+	addition
-	substraction	*	multiplication
/	division		

Arithmetic operations are done in floating point. If the result is integral, the result string will be integer. A logical operator will give a character 0 (zero), if the result is FALSE, and a character 1 (one), if the result is TRUE.

Expressions

Script expression consists of operands, operatores and parentheses.
The precedence of the operators is

```
-  !(unary)
/  *
+  -
%
=  !=  >  >=  <  <=
&
|
```

Within the same precedence level, operations are performed left to right. Parentheses modify the order of operation in the expected ways.

To concatenate two or more strings using the concatenate operator (%) or just two single quotes (' ') instead of the operator.

Example:

```
col1 = '16  17  18  19  20  '
col2 = '21  22  23  24  25  '
col3 = '26  27  28  29  30'
colors = col1%col2%col3
or  colors = col1''col2''col3

'set ccols 'colors
is equal to  'set ccols 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30'
```

Standard input/output

To write information to the terminal (standard output):

say expression

To write an input request string:

prompt expression

The 'prompt' command works the same way as 'say' except it does **not append** a carriage return!

To read an input string/value from the standard input:

pull variable

The script pauses for the user keyboard input (up to the carriage return), and the string entered by the user is assigned to the indicated variable name.

Examples:

```
line = 'Peter Pan, the flying one'
say line

prompt 'Enter latitude:  '
pull lat
prompt 'Enter longitude: '
pull lon
'set lat 'lat
'set lon 'lon
```

To combine variables and comments writing to standard out:

```
say 'She said, it is 'line
```

produces

```
She said, it is Peter Pan, the flying one
```

Control flow

IF Block:

```
if  expression          each must be seperated lines
   command
   .....
else                    optional
   command
   .....
endif                   required!
```

Note: There is NO 'else if ' element implemented in GrADS!

Example:

```
if(i=10);  j=20; endif
```

is equal to

```
if(i=10)
   j=20
endif
```

WHILE Loop:

```
while expression
   command
   .....
endwhile
```

To continue the while loop use the **continue** command; to exit the while loop use the **break** command

Example:

```
t=1
while(t<10)
   'set  t  't
   'd  z'
   t = t + 1
endwhile
```

Functions

Functions are invoked as a script expression is beeing evaluated. Functions always have a single string result, but may have one or more string arguments! Functions are invoked by:

name(arg1, arg2, arg3, ..., argn)

If the function has no arguments, you must still provide the parentheses:

name()

To define a user own function by using the function definition record:

function name(var1, var2, var3, ..., varn)

To return from a function, use the return command:

return(expression)

The expression is optional, if not provided, a NULL string will be returned.

Example:

```
x = 10
y = 30
z = addit(x,y)
say 'Result of addition:  z = 'z
....
function addit(var1,var2)
   sum=var1+var2
return (sum)
```

terminal output

```
Result of addition: z = 40
```

Sending commands

The statement record consists only of an expression

expression

The expression is evaluated, and the resulting string is submitted to GrADS as a command. After this record is executed, the script variable ‘**result**’ is given the value.

Examples:

```
hallo = ‘draw string 4.0 8.0 HALLO‘
hallo
```

or

```
‘query‘
say result
```

produces

```
GrADS Version 1.7Beta6 ---
LATS=GRIB_NCSA_HDF__SDF_READ=NCSA_netCDF_HDF
query or q Options:
 q config   List configuration of this build
 q files:   Lists open files
 q file n:  Gives info on particular file
 q define:  Lists currently defined variables
 q fwrite:  Fwrite Status
 q lats:    State of the GrADS-LATS Interface
 q dims:    Gives current dimension environment
 q time:    Gives current time
 q gxinfo:  Gives graphics environment info
 q shades:  Gives colors and levels of shaded contours
 q pos:     Waits for mouse click, returns position
 q w2gr:    Convert world to grid coordinates
 q gr2w:    Convert grid to world coordinates
 q w2xy:    Convert world to x,y screen coordinates
 q xy2w:    Convert x,y screen coordinates to world
            coordinates
 q gr2xy:   Convert grid to x,y screen coordinates
 q xy2gr:   Convert x,y screen to grid coordinates
 q pp2xy:   Convert pre-projected grid coordinates to
            screen x,y coordinates
 q defval:  Gives defined grid value given grid i,j
```

Intrinsic functions

To get a single word from a string:

res = subwrd(string,word)

The result is the nth ‘word’ from the string. If string is too short, the result is the NULL string. ‘word’ must be an integer value.

To get a single line from a string containing several lines:

res = sublin(string,line)

The result is the nth ‘line’ from the string. If the string has too few lines, the NULL string is returned. ‘line’ must be an integer value.

To get a part of a string:

res = substr(string,start,length)

The substring of ‘string’ starting at location ‘start’ for length ‘length’ will be returned. If the string is too short, the result will be short or the NULL string. ‘start’ and ‘length’ must be an integer value.

Examples:

```
‘query time‘
res = subwrd(result,3)
year = substr(res,9,4)
say year
```

produce e.g. 1880

The function sublin is very usefull, if you want to control opening, reading, writing and closing an extern ASCII file.
For example, the first record in the ASCII file ‘the_title.txt’ to be read is

Szenario A 1880 - 2099

The following part of a script will open, read and close the file, controling the status of each statement:

```
ret = read(‘the_title.txt’)
code = sublin(ret,1)
if(code != 0)
   say ‘read error  #‘code
   ‘quit‘
endif
title = sublin(ret,2)
‘draw title ‘title
ret = close(‘the_title.txt’)
code = sublin(ret,1)
if(code != 0)
   say ‘close error #’code
   ‘quit‘
endif
```

I/O functions

To read records from an ASCII file:

res = read(filename)

The result is a string containing two lines. The first line is the return status and the second line is the record. The record may have a maximum of 80 characters. Use the sublin intrinsic function to seperate the lines.

The return status of read:

0	ok
1	open error
2	end of file
8	file open for write
9	I/O error

To write records to an ASCII output file:

res = write(filename, record <,append>)

The record is written to the file ‘filename’. On the first call to write for a particular file, the file is opened in write mode; this will destroy an existing file ‘filename’! If you use the optional append flag, the file will be opened in append mode, and all writes will be appended to the end of the file.

The return status of write:

0	ok
1	open error
8	file open for read

To close an opened file:

res = close(filename)

The close command closes the named ASCII file and can also be used to rewind the file.

The return status of close:

0	ok
1	file not open

Examples:

```
‘q file 1‘
ret = result
res = write(‘file_1.txt’, ret)
status = sublin(res,1)
if(status != 0)
   say ‘write error  #‘status
   ‘quit‘
endif
res = close(‘file_1.txt’)
status = sublin(res,1)
if(status != 0)
   say ‘close error #’status
   ‘quit‘
endif
```

Example script

The following example script draws 1200 shaded contour frames (1200 time records). The year, which will be used in the title string, is read from the ‘query time’ result. The private colors are defined in the function palette(). The ‘set clip ..’ command is used with the ‘set dbuff on’ and ‘swap’ commands to restrict the redraw of the plot to areas with changes from frame to frame.

At the DKRZ - Hamburg, videos were recorded using this kind of animation within GrADS. To achieve smooth animations, the single frame technique had been applied.

```
‘reinit‘
‘open descriptor.ctl‘
count = 0
rec = 1200
incr = 1;   t = 1
palette()
‘set vpage 0.0 11.0 0.0 8.5‘
‘set parea 1.0 10.0 1.4 7.9‘
‘set dbuff on‘
‘set mpdset lowres‘
‘set map 0 1 10‘
‘set lat -90 90‘
‘set lon -180 180‘
‘set mpvals -180 180 -90 90‘
‘set mproj robinson‘
‘set grid on 5 0‘
while (count < rec)
            ‘set t  ‘t
            ‘q time‘
            res = subwrd(result,3)
            year = substr(res,9,4)
            ‘set grads off‘
            ‘set string 1 c 8‘
            ‘set strsiz 0.23 0.26‘
            ‘draw string 5.5 7.6 Aerosol - Control ‘year
            ‘set gxout shaded‘
            ‘set cint 1.0‘
            ‘set cmin -4.0‘
            ’set cmax 4.0‘
            ‘set clevs -4.0 -3.0 -2.0 -1.0 0.0 1.0 2.0 3.0 4.0‘
            ‘set ccols  17   18   19   21   22  23  24  25  26  27‘
                  ‘display data‘
            ‘set gxout contour‘
            ‘set cterp off‘
            ‘set csmooth off‘
            ‘set cint 1.0‘
            ‘set clab off‘
                  ‘display data‘
            ‘run cbar.gs‘
            ‘set clip 1.0 10.0 1.4 7.9‘
            ‘swap‘
       count = count + incr
       t = t + incr
endwhile

function palette()
       ‘set rgb  16  0  0  20‘
       ‘set rgb  17  0  29  85‘
       ‘set rgb  18  0  44  128‘
       ‘set rgb  19  0  83  230‘
       ‘set rgb  21  0  151  250‘
       ‘set rgb  22  104  173  255‘
       ‘set rgb  23  177  213  255‘
       ‘set rgb  24  255  250  110‘
       ‘set rgb  25  255  209  116‘
       ‘set rgb  26  255  160  80‘
       ‘set rgb  27  255  100  65‘
return
```