

# The Macro Language

- Enhancements with Release 9
- Advantages of using compound macro statements
- Adding comments to the macro file
- Frequent errors when writing macros
- Macro writing tips

*Presented by*

*Diane Larin, INRO Consultants Inc.*

*13th Annual International EMME/2 Users' Conference – Houston, October 1998*

# Enhancements to macro language with Release 9

- Access to 32 macro parameters (previously 9)
- New command (~#)
- Access to substrings using %tx\_N% or %tx\_-N% (same as %tx.N% or %tx.-N%)
- New text registers tc
- Field width with explicit left/right justification
- Access to name and description of any matrix (not only ms)
- Access to matrix time stamps
- Easier access to external files
- More information available through register p
  - 2012 - 2018 Access to various dimensions of the data bank
  - 2021 - 2033 Access to number of errors for various invalid operations (division by zero, square root of a negative number ...)

# New text registers

- Read-only
- Available registers
  - **tp** project title
  - **ts** scenario title
  - **tl** unit of length
  - **tc** unit of cost
  - **te** unit of energy
  - **tu** user initials (similar to **%u%**)

Example:

```
~+;~t0=%ts%;~t1=%%1%%%;~?!t1=KILDONAN  
~<kildonan.mac
```

# Field width with explicit left/right justification

- `%reg_>W%`: any substitution (numeric or text) **right** justified in  $W$  characters
- `%reg_<W%`: any substitution (numeric or text) **left** justified in  $W$  characters

Example:

---

```
%t1%: %x%          modes: 4
%t2%: %y%          regular nodes: 905
%t3%: %z%          directional links: 2981
```

---

```
%t1_20%: %x_5%    modes           :      4
%t2_20%: %y_5%    regular nodes       :    905
%t3_20%: %z_5%    directional links    :  2981
```

---

```
%t1_>20%: %x_<5%          modes: 4
%t2_>20%: %y_<5%          regular nodes: 905
%t3_>20%: %z_<5%          directional links: 2981
```

---

# Access to matrix information

- Name: `%mTx.n%`
- Description: `%mTx.d%`
- Time stamp: `%mTx.t%`
- $T = \mathbf{f}, \mathbf{o}, \mathbf{d}$  or  $\mathbf{s}$
- $x = 1 \dots 99, \mathbf{x}, \mathbf{y}$  or  $\mathbf{z}$

Example: checking for the existence of a matrix before deleting it

```
~+;~t1=%1%.t%  
~t1=%t1.1%  
~?!t1=%  
~+;3.12;2;%1%;y;q
```

# Access to external files

- **Append** command: *~>>filename*
  - equivalent to *~>filename* if *filename* does not exist
  - *filename* is opened and positioned at the end of the file
  - all following commands are appended to *filename* until the next *~>* or *~>>*
- **Quoted output** command: *~"*
  - no immediate effect on the macro currently running
  - used in combination with *~>filename* or *~>>filename* commands
  - **only** the characters following the leading *~"* are written to *filename*
- **Open read file** command: *~<<filename*

Remains open until

  - an empty open macro file command is executed (*~<<*)
  - a new file is opened (*~<<newfilename*)
  - a file read command *~@* fails
  - the macro that opened the file is terminated
- **File read** command: *~@*
  - reads one line from the file previously opened with *~<<*
  - similar to *~\**
  - EOF or I/O error can be tested using *~?e*
  - each macro level can have its own separate file open for reading

## Ex. 1: Save the values of the `rx` registers in an external file.

```
~x=0  
~+;~x+1;~>>rxval;~"%%x%% %%rx%%;~>>;~?x<250;~$
```

Contents of result file `rxval`

```
1 10  
2 35669244  
3 10485  
4 5060  
5 89  
6 283091392  
7 774794560  
8 6  
9 15  
10 1  
...
```

## Ex. 2: Use `g` registers to create a “look-up table” using values stored in an external file.

```
~#lookup: fill gx registers with values in field %2% of file %1%
~<<%1%
~#keep value of second parameter
~x=%2%
~z=0
~+;~t0=~@;~?e;~$>END_OF_FILE;~z+1;~gz=%x%;~/g%%z%%=%%gz%%;~?z<250;~$
~:END_OF_FILE
```

### Contents of input file `lktab`

```
1    2000 10000 10000 1
10   1200 10555 10555 3
14   800  10211 10211 7
15   200  10244 10244 10
19   400  10338 10004 2
```

### Output when calling macro

```
Enter: Next module=~<lookup lktab 4
<~/g1=10000
<~/g2=10555
<~/g3=10211
<~/g4=10244
<~/g5=10004
<~<
```

# Advantages of using compound macro statements

Use compound commands for:

- Implementing small loops

```
~+;~x+1;~?!x>%1%;~$
```

- Enhancing readability (answers forming a logical entity are grouped)

```
~#compute passenger miles  
~+;1;y;tmpt3;len*voltr ; ;4;all;all;2;~?q=2;2
```

- Applying a condition to several logical lines

```
~?e  
~+;~/Error XXX;~$>END
```

- Implementing double level of substitution

```
~+;~t1=%%1%.t%
```

# Adding comments to the macro file

- Many possibilities
  - `~#` to comment the macro file **without** copying the comments to the terminal
  - `~/` to copy the comments to the terminal (mostly useful in “silent” mode)
  - `/comments` at the end of a line (copied to the screen unless in “silent” mode)
- Good practice to describe the expected parameters at the beginning of a macro

```
~#trassign <boa_t> <wait_f> <wait_w> <aux_w> <boa_w>
~#perform transit assignment with the following parameters:
~#   boa_t:  boarding time (in minutes)
~#   wait_f: wait time factor
~#   wait_w: wait time weight
~#   aux_w:  aux. time weight
~#   boa_w:  boarding time weight
~#
~x=%0%
~?!x=5
~+;~/Usage: trassign <boa_t> <wait_f> <wait_w> <aux_w> <boa_w>;~$>END
~/performing transit assignment with boa_t=%1% wait_f=%2% wait_w=%3% aux_w=%4% boa_w=%5%
```

- Add logbook comments at the beginning and end of the macro  
Example: `c='macroname %1% %2% %3%'`
- Use the menu command `m=` to call a module

# Frequent errors when writing macros

- Invalid assignment commands.

Example: what is the output from the following commands?

```
~x=1
~y=2
~x=%y%+3
~/x=%x%
```

- Premature substitution in compound commands.

Example: what is the output from the following commands?

```
~+ ; ~x=1 ; ~y=5
~+ ; ~x+1 ; ~x*%y% ; ~/x=%x%
```

Substitutions are performed in 2 steps:

- once when the compound command is **read**
- once when the subcommand is **executed**

- Using substitution on the left side of an expression

Example:

```
~?%0%=0
~/Missing argument
```

- Using `t0` for accessing a specific word in a string before saving its contents

- Strings too long following one or more substitutions
  - maximum length of a macro line: 128 characters
  - project and scenario title: 60 characters
  - matrix description: 40 characters

Example:

```
~#keep first 40 characters of macro parameters
~#to form matrix description
%t0_40%
...
```

- Choice of floating point registers: **gx** or **rx**
  - changes to **rx** are **local**, changes to **gx** are **global**
  - **gx** are initialized to zero at the beginning of each module
  - modifying **gx** changes the value in the  $x$ -th slot in the temporary stack
  - **gx** values can be copied to **rx** registers before leaving the module

For example:

- Use **gx** registers to initialize a “lookup” table
- Use **rx** registers to perform floating point calculations

# Macro writing tips

- Verify the presence of all mandatory parameters (**~x=%0%;~?!x=**)

- Test if a scenario is assigned before calling a module from group 6

Example:

```
~+;~y=%f%;~y&2048;~?y=0
~+;~/Scenario with no auto assignment;~$>STOP
```

- When using an extra attribute, make sure **in the macro** that the attribute exists and is correctly initialized

Example:

```
m=2.42
~+;2;3;@lsymb;~?!e;line symbol index;
q
m=2.41
...
```

- Test the availability of a scenario by using the **p** register.

Example:

```
m=1.22
~+;~#delete 3rd scenario if already defined;~p=103;~?!p=0
~+;2;%p%;y
~+;~#copy scenario;3;1000;3000;TOLL ON BRIDGE 304-412;y
q
```

- Test if the macro is called from the appropriate module (**~?m=**)
- Verify the presence of an optional question (**~?q**)
- Suppress pauses (**~o|32**) to speed up the macro
- Use menu commands **reports=xxx, plots=xxx** ... to create result files specific to the macro (**reports=^, plots= ^...** at the end)
- Set register **o** by using bitwise OR (**|**)

- Set a macro in silent mode only if switch 15 is off

Example:

```
~+;~?!i&32768;~o|7
```

- Write macros independent of particular switch settings

Keep the value of the switch bit pattern (**%i%**) before modifying a switch

Example:

```
~x=%i%
on=0
off=2
~?!x&1
~+;~/switch 0 was off;off=0
~?x&4
~+;~/switch 2 was on;on=2
```

- Write macros independent of particular module settings  
Define the module parameters before generating a report or a plot (do not save on disk)
- Reserve scalar matrices **ms90** - **ms99** for temporary data
- Store temporary data vectors in temporary network attributes (Module 2.41)  
or extra attributes with names beginning with **@tmp**.

- Be careful about operating system dependency

Example:

```

~p=2004    / find operating system
~?p=1      / UNIX
~t1=%t1%/
~?p=2      / DOS
~t1=%t1%\
batchin=' %t1%d%%m%%.in '

```

- Choose the terminal type wisely (graphic or not)
- Use the report command **c** when sending report to the terminal
- Use **quit** to leave a module