

# GAMS Cheat Sheet

## Declarations

GAMS objects have to be declared before their first use. Main objects are

<b>set</b>	Collection of elements used for indexing. $S = \{a, b, c\}$ is written in GAMS as <code>SET S / a, b, c /;</code> . A sequence of elements, such as <code>t=1990:2010</code> , can be entered as <code>SET t 'Year' / 1900*2010 /;</code> , where <code>'Year'</code> is an optional explanatory text.
<b>parameter, scalar, table variable</b>	Exogenous parameters to be entered or calculated by the modeler. Endogenous variables to be determined by GAMS. It is possible to enter the following prefixes before <b>variable</b> to specify the variable type: <b>positive</b> , <b>negative</b> , <b>binary</b> (variable is 0 or 1), <b>integer</b> .
<b>equation</b>	Symbolic equations.
<b>model</b>	Collection of equations. To declare a model that includes all the equations: <code>model model_name / all /;</code> . To include a list of equations: <code>model model_name / eq_name1, eq_name2 /;</code>

## Data entry

The general expression for declaring and initializing parameters is

*parameter-type parameter\_name [parameter\_description] [/ parameter\_value /];*

Examples:

```
scalar rho "discount rate" / .15 /;
parameter b(i) / seattle 20, san-diego 45 /
           salaries(employee,manager,department)
           /anderson .murphy .toy      = 6000
             hendry .smith .toy       = 9000
           hoffman .morgan .cosmetics = 8000 / ;
```

## Variable attributes

To each variable is associated a series of attributes:

- .l Level of the variable. Receives new values when a model is solved.
- .lo Lower bound (default to `-inf`).
- .up Upper bound (default to `inf`).
- .fx To fix a variable (set in one command the same value to level, lower and upper bounds):  
`x.fx(i) = 1;`
- .m Marginal (or dual) value. Receives new values when a model is solved.

## Arithmetic and functions

Arithmetic operations:

`+`, `-`, `*`, `/`, `**` (exponentiation, `x**y` is defined only for  $x \geq 0$ , if  $x$  can be negative, use `power(x,y)` instead).

Most common functions:

`abs()`, `cos()`, `exp()`, `log()`, `log10()`, `max(,...)`, `min(,...)`, `power(,)`, `round()`, `sin()`.

Relationship operators:

`lt`, `<`, `le`, `<=`, `eq`, `=`, `ne`, `<>`, `ge`, `>=`, `gt`, `>`.

Logical operators:

`not`, `and`, `or`, `xor`.

Special symbols:

`inf` Plus infinity.

`-inf` Minus infinity.

`na` Not available, used for missing data.

`undef` Undefined, result of an undefined operation such as `3/0`.

`eps` Numerically equal to zero, but considered as existing. For example, `sum(i$z(i),1)` equals 0 if `z(i)=0` and `sum(i$z(i),1)` equals `card(i)` if `z(i)=eps`.

## Conditional expressions with dollar condition

Logical expression can be expressed with a dollar condition. For example:

`a$(b > 1.5) = 2;` means if  $b$  is greater than 1.5 then  $a$  equals 2. If  $b$  is less than-or-equal to 1.5 then the value of  $a$  remains unchanged.

It can also be used on the right hand side. For example:

`a = 2$(b > 1.5);` means that  $a$  equals 2 if  $b$  is greater than 1.5, else  $a$  equals 0.

## Indexing

### Basic indexing

<code>x(i) = 12;</code>	Assign all elements of $x$ to 12.
<code>b('seattle') = 20;</code>	Assign the element <code>seattle</code> of $b$ to 20.
<code>sum(i,x(i))</code>	Sum $x$ over the set $i$ : $\sum_i x_i$ .
<code>sum((i,j),x(i,j))</code>	Sum $x$ over the sets $i$ and $j$ : $\sum_{i,j} x_{i,j}$ .
<code>prod(j,y(i,j))</code>	Multiply $y$ over the set $j$ : $\prod_j y_{i,j}$ .
<code>alias(i,j)</code>	Declare that the set $j$ can be used instead of $i$ .
<code>y = smax(i,x(i));</code> or <code>y = smin(i,x(i));</code>	Find the largest or smallest value of a symbol indexed over a set.

### Advanced indexing

On ordered sets (for example one defined by `SET t 'Year' / 1900*2010 /;`):

<code>ord(t)</code>	Returns the position of a member in a set: <code>parameter val(t);</code> <code>val(t) = ord(t);</code> Here <code>val('1900')</code> will be 1, <code>val('1909')</code> 10, and <code>val('2010')</code> 111.
<code>card(t)</code>	Returns the number of elements in a set: <code>card(t)</code> will return 111.
lags and leads	It is possible to use lag or lead operators on ordered sets. For example an equation defining the evolution of capital stock would be: <code>eq.k(t+1).. k(t+1) =e= (1-delta)*k(t) + i(t);</code>

`sameas(r,s)` can be used to test if the active elements of  $r$  and  $s$  are the same. For example:

`a(r,s)$(not sameas(r,s)) = 10;` would assign 10 to all non-diagonal elements of  $a$ .

It is possible to define subsets: sets whose members must all be members of some larger sets. For example:

```
set
  i "all sectors" / light-ind, food+agr, heavy-ind, services /
  t(i) "traded sectors" / light-ind, food+agr, heavy-ind /;
```

The assignment can also be made dynamically (the set is then called a dynamic set):

```
set j(i);           Declare j as a subset of i.
j(i) = yes;         Assign all elements of i to j.
j('light-ind') = no; Remove the element 'light-ind' from j.
Or alternatively: j(i)$(not sameas(i,'light-ind')) = yes;.
```

Dynamic subsets present the following restrictions: it is not possible to declare variables defined on dynamic subsets; and they are not ordered, even if their parent sets are.

## Equation definition

An equation named *eqname* is defined by

*eqname(index) .. expression eq\_type expression ;*

Main equation types (*eq\_type*):

`=e=` Equality: rhs must equal lhs.

`=g=` Greater than: lhs must be greater than or equal to rhs.

`=l=` Less than: lhs must be less than or equal to rhs.

Solve statement

solve *model\_name* using *model\_type* (maximizing|minimizing *objective\_name*)

Main model types (*model\_type*):

- cns Constrained Nonlinear System: square system of nonlinear equations,  $f(x) = 0$ .
- lp Linear programming: optimization problem with linear objective and constraints.
- mcp Mixed Complementarity Problem.
- nlp Nonlinear programming: optimization problem with nonlinear objective and constraints.
- qcp Quadratic constraint programming: optimization problem with quadratic objective and constraints.

Display

display *x*, *y*.1; to ask GAMS to write in the listing file (file with the .lst extension) the value of *x* and *y*. For variables, one has to precise the attribute (.1 here).

option decimals = *N* to restrict the display to the first *N* decimals.

Flow control

GAMS contains 3 types of loops:

for to loop over a parameter:

```
scalar i;
for(i = 1 to 1000 by 10,
  display i;
);
```

loop to loop over a set:

```
loop(t, pop(t+1) = pop(t) + growth(t));
```

while to loop over a general condition:

```
scalar x / 0 /;
while(x <= 10,
  x = x + 1;
);
```

Use of the if-else statement:

```
if(x <= 0,
  y = 1;
elseif(x > 0 and x < 1),
  y = 2;
else
  y = 3;
);
```

To stop GAMS if a condition is met use abort:

```
abort$(abs(residuals) > 1E-6) "Residual not null", residuals;
```

Dollar control

Dollar control options can alter GAMS behavior in several ways. The \$ symbol must always be placed in the first column. They are executed at compile time, so before any calculation take place. Most important dollar control options:

- \$exit GAMS stop reading the file after \$exit.
- \$include Use \$include filename to insert the contents of the file.
- \$ontext/\$offtext Use to enclose severals lines of comments.
- \$set Use \$set varname varvalue to define an environmental variable, which can be called later using %varname%.

Options

Some options can be set using the following syntax:

```
option option_name = option_value;
```

Main options:

option_name	Default	Interpretation
decimals	3	Number of decimals printed.
iterlim	1000	Limit on the number of iterations used to solve a model.
limcol	3	Control the number of columns (variables) listed at each solve.
limrow	3	Control the number of rows (equations) listed at each solve.
reslim	1000	Limit on the units of processor time used to solve a model.
solver (cns, nlp, lp, ...)	Installation default	Control the solver used to solve a particular model type.

Example: option limcol = 0;

Comments

A line starting with an asterisk '\*' is commented:

```
* This line is a comment
```

To comment several lines, it is possible to place them between a pair of \$ontext/\$offtext:

```
$ontext
Any lines between $ontext and $offtext are commented
$offtext
```

End-of-line comments can be enabled using \$eolcom followed by the chosen special character:

```
$eolcom #
x = 1; # This is an end-of-line comment
```

In-line comments can be enabled using \$inlinecom followed by a pair of one or two character sequence (default to /\* \*/):

```
$inlinecom { }
x { This is an in-line comment } = 1;
```

GDX files

A GDX file is a binary file that can contains information on sets, parameters, variables, and equations. GDX files are very useful to enter data, to explore results, and to import/export data from various file formats (e.g., csv, Excel, ...).

Compile phase (before any calculation)

```
$gdxin file_name.gdx      Open the GDX file for reading.
$load id1 id2=gdxid2      Read symbols id1 and gdxid2 from the GDX file and assign them to id1 and id2 that have been previously declared.

$gdxin                    Close the GDX file currently open.
Same thing with $gdxout and $unload to write data to a GDX file during the compile phase.
```

Execution phase (after calculations)

```
execute_load 'file_name.gdx' id1, id2=gdxid2      Read symbols id1 and gdxid2 from the GDX file and assign them to id1 and id2 that have been previously declared.

execute_unload 'file_name.gdx' id1, id2=gdxid2      Write to the GDX file the symbols id1 and id2 and assign id2 to the symbol gdxid2.
```