GAMS GDX facilities and tools

GAMS Development Corporation

11/25/2014

© 2014 GAMS Development Corporation
# Table of Contents

## Part I Introduction

### 1 Compile Phase

- Compile Phase Reading Data ................................................................. 1
  - Compile Phase Example 1 ................................................................. 2
  - Compile Phase Example 2 ................................................................. 3
  - Compile Phase Example 3 ................................................................. 4
  - Compile Phase Example 4 ................................................................. 4

- Compile Phase Writing Data ............................................................... 5

## Part II Using the GDX facilities in GAMS

### 2 Execution phase

- Execution Phase Writing Data ........................................................... 7
  - Indexed writing and reading of data .............................................. 8

## Part III Inspecting a GDX file

## Part IV GDX utilities

## Part V GDXXRW

### 1 GDXXRW parameters and options

- GDXXRW Options (Immediate) ............................................................ 12
- GDXXRW Symbols ............................................................................ 15
- GDXXRW Options ............................................................................ 18
- Syntax elements ............................................................................. 20
- GDXXRW Ranges ............................................................................. 20

### 2 GDXXRW Warning

### 3 GDXXRW examples

- Read spreadsheet: Example 5 ............................................................ 22
- Read spreadsheet: Example 6 ............................................................ 22
- Read spreadsheet: Example 7 ............................................................ 23
- Read spreadsheet: Example 8 ............................................................ 23
- Read spreadsheet: Example 9 ............................................................ 23
- Read spreadsheet: Example 10 ......................................................... 24
- Write to spreadsheet: Example 11 ................................................... 24
- Write spreadsheet: Example 12 ........................................................ 25
- Read / Write spreadsheet: Example 13 .......................................... 25
- Read / Write spreadsheet: Example 14 .......................................... 26
- Write spreadsheet using a filter: Example 15 .................................. 29
- Write spreadsheet using text and hyperlinks .................................... 29
- CMerge Example ............................................................................ 30
- CMerge Example ............................................................................ 31

## Part VI GDXDUMP

### 1 GDXDUMP Example 16 ................................................................. 33
<table>
<thead>
<tr>
<th>Part</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VII</td>
<td>GDXDIFF</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>GDXDIFF Example 17</td>
<td>36</td>
</tr>
<tr>
<td>VIII</td>
<td>GDXMERGE</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>GDXMERGE Example 18</td>
<td>37</td>
</tr>
<tr>
<td>IX</td>
<td>GDXRANK</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>GDXRANK Example 19</td>
<td>38</td>
</tr>
<tr>
<td>X</td>
<td>GDXRENAME</td>
<td>39</td>
</tr>
<tr>
<td>XI</td>
<td>GDXCOPY</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>GDXCOPY Example 20</td>
<td>41</td>
</tr>
<tr>
<td>XII</td>
<td>CSV2GDX</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>CSV2GDX Example 21</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>CSV2GDX Example 22</td>
<td>44</td>
</tr>
<tr>
<td>XIII</td>
<td>XLSDump</td>
<td>45</td>
</tr>
<tr>
<td>XIV</td>
<td>XLSTalk</td>
<td>46</td>
</tr>
<tr>
<td>XV</td>
<td>INVERT</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Invert Example 21</td>
<td>48</td>
</tr>
<tr>
<td>XVI</td>
<td>CHOLESKY</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Cholesky Example 22</td>
<td>49</td>
</tr>
<tr>
<td>XVII</td>
<td>EIGENVALUE</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>EigenValue Example 23</td>
<td>50</td>
</tr>
<tr>
<td>XVIII</td>
<td>EIGENVECTOR</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>EigenVector Example 24</td>
<td>51</td>
</tr>
<tr>
<td>XIX</td>
<td>MCFilter</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>MCFilter Example</td>
<td>52</td>
</tr>
</tbody>
</table>
1 Introduction

This document describes the GDX (GAMS Data Exchange) facilities available in GAMS. In addition to these facilities, there are a few utilities to work with GDX files.

A GDX file is a file that stores the values of one or more GAMS symbols such as sets, parameters variables and equations. GDX files can be used to prepare data for a GAMS model, present results of a GAMS model, store results of the same model using different parameters etc. A GDX file does not store a model formulation or executable statements.

GDX files are binary files that are portable between different platforms. They are written using the byte ordering native to the hardware platform they are created on, but can be read on a platform using a different byte ordering.

Users can also write their own programs using GDX files by using the gdxdclib library. The interface and usage for this library is described in a separate document; see apifiles\gdx\gdxioapi.chm

Compression:
Starting with version 22.3 of GAMS, gdx files can be written in a compressed format. Compression is controlled by the environment variable GDXCOMPRESS. A value of 1 indicates compression.

GDX files can be converted to a compressed format or an older format too; see GDXCOPY

2 Using the GDX facilities in GAMS

Reading and writing of GDX files in a GAMS model can be done during the compile phase or the execution phase. A GDX file can also be written as the final step of GAMS compile or execute sequence.

The GAMSIDE can read a GDX file and display its contents.

2.1 Compile Phase

During compilation, we can use dollar control options to specify the gdx file and the symbols to read or write. Reading during the compilation phase also allows us to define the elements of a set and the subsequent use of such a set as a domain.

2.1.1 Compile Phase Reading Data

The following directives are available for reading data from a GDX file into GAMS during compilation of a GAMS model:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GDXIN</td>
<td></td>
<td>Close the current GDX input file</td>
</tr>
</tbody>
</table>
Using the GDX facilities in GAMS

<table>
<thead>
<tr>
<th>Filename</th>
<th>Specify the GDX file to be used for reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LOAD</td>
<td>List all symbols in the GDX file</td>
</tr>
<tr>
<td>$LOADDC</td>
<td></td>
</tr>
<tr>
<td>id1 id2 ... idn</td>
<td>Read GAMS symbols id1, id2, ... idn from the GDX file</td>
</tr>
<tr>
<td>id1=gdxid1 id2=gdxid2</td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file</td>
</tr>
<tr>
<td>id1&lt;gdxid1 id2&lt;gdxid2.dim3 id1&lt;=gdxid1 id2&lt;=gdxid2.dim3</td>
<td>Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (&lt;) or from the left (&lt;=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the fourth example.</td>
</tr>
<tr>
<td>$LOAD</td>
<td>id=*</td>
</tr>
<tr>
<td></td>
<td>Loads all unique elements from the gdx file into set id</td>
</tr>
<tr>
<td>$LOADM</td>
<td></td>
</tr>
<tr>
<td>$LOADR</td>
<td></td>
</tr>
<tr>
<td>$LOADDCM</td>
<td></td>
</tr>
<tr>
<td>$LOADDCR</td>
<td></td>
</tr>
<tr>
<td>$LOADIDX</td>
<td>id1 id2 ... idn</td>
</tr>
<tr>
<td></td>
<td>Read GAMS symbols id1 id2 ... idn from the GDX file. Each symbol should have been written using an indexed write; see Execute_UnloadIDX.</td>
</tr>
<tr>
<td></td>
<td>id1=gdxid1 id2=gdxid2</td>
</tr>
<tr>
<td></td>
<td>Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file. Each symbol should have been written using an indexed write; see Execute_UnloadIDX.</td>
</tr>
</tbody>
</table>

Notes:

- Only one GDX file can be open at the same time.
- When reading data, the symbol to be read has to be defined in GAMS already

2.1.1 Compile Phase Example 1

The transport.gms model has been modified to use the demand data from an external source. Only the relevant declarations are shown.

The parameter B is read from the GDX file using the name 'demand', and only those elements that are in the domain J will be used. Values for parameter B that are outside the domain J will be ignored without generating any error messages.
*Example 1
Set
  j markets / new-york, chicago, topeka / ;
Parameter
  B(j) demand at market j in cases ;
$GDXIN demanddata.gdx
$LOAD b=demand
$GDXIN

2.1.1.2 Compile Phase Example 2

In this example, the set J is also read from the GDX file, and is used as the domain for parameter B. All elements read for the set J will be used. Values for the parameter B that are outside the domain J will be ignored. Note that the dimension of set J is set to one by specifying its domain.

*Example 2
$GDXIN demanddata.gdx
Set
  J(*) markets;
$LOAD j=markets
Parameter
  B(j) demand at market j in cases ;
$LOAD b=demand
$GDXIN

2.1.1.3 Compile Phase Example 3

Using $LOAD to get a listing of all symbols

*Example 3
$GDXIN transport.gdx
$LOAD

Writes the following to the listing file:

Content of GDX C:\XLSFUN\TRANSPORT.GDX

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Dim</th>
<th>Count</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set</td>
<td>1</td>
<td>2</td>
<td>i</td>
</tr>
<tr>
<td>2</td>
<td>Set</td>
<td>1</td>
<td>3</td>
<td>j</td>
</tr>
<tr>
<td>3</td>
<td>Parameter</td>
<td>1</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>Parameter</td>
<td>1</td>
<td>3</td>
<td>b</td>
</tr>
<tr>
<td>5</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>d</td>
</tr>
<tr>
<td>6</td>
<td>Parameter</td>
<td>0</td>
<td>1</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>Parameter</td>
<td>2</td>
<td>6</td>
<td>c</td>
</tr>
<tr>
<td>8</td>
<td>Variable</td>
<td>2</td>
<td>6</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>Variable</td>
<td>0</td>
<td>1</td>
<td>z</td>
</tr>
<tr>
<td>10</td>
<td>Equation</td>
<td>0</td>
<td>1</td>
<td>cost</td>
</tr>
<tr>
<td>11</td>
<td>Equation</td>
<td>1</td>
<td>2</td>
<td>supply</td>
</tr>
<tr>
<td>12</td>
<td>Equation</td>
<td>1</td>
<td>3</td>
<td>demand</td>
</tr>
</tbody>
</table>
2.1.1.4 Compile Phase Example 4

Sometimes, a set is implicitly given by the elements of a parameter symbol. For example,

```
parameter a(i) / seattle 350, san-diego 600 /
```

in `transport.gms` implicitly defines the set of plants \( i \). GAMS does not allow us to provide domain checked data, if the data for domain sets is unknown. So this code produces a compilation error:

```
Set i plant;
parameter a(i) capacity / seattle 350, san-diego 600 /;
```

When entering data directly in the GAMS source adding the domain sets before the actual parameter declarations is usually not a problem, but when data comes from external sources (e.g. spreadsheets, databases, etc), this often results in an additional query to the database, spreadsheet etc. Nowadays, such data exchange happens mostly via the GD facility. With the domain load capability of the compile time load instructions ($load, $loadDC, $loadR, $loadM, $loadDCM, and $loadDCR) one can project an index position from a parameter or set symbol in the GDX container and load this slice into a one dimensional set. Here is a simple example:

```
Set i plant;
Parameter a(i) capacity;
$gdxin data
$load i<adata a=adata
```

This will try to load set elements from the GDX parameter symbol `adata` into the set \( i \) and next load the GDX parameter `adata` into the GAMS parameter `a`. The latter one is no problem anymore, since the data for set \( i \) is known when loading symbol `a`. GAMS will use the domain information stored in GDX of parameter `adata` to identify the index position to project on. If no appropriate domain information can be found in GDX, the GAMS compiler will generate an error. In such case the user can explicitly select an index position (here first index position) from the GDX symbol:

```
$load i<adata.dim1 a=adata
```

The automatic index position matching (i.e. no `.dimN`) using the domain information stored in GDX matches on the name of the set to be loaded and the domain set names stored in GDX for the symbol. The domain in GDX are searched from right to left (start with \( n \)=symbol dimension, then \( n-1, n-2, \ldots \)) and stops at the first match. With the projection symbol `<=`, the domain in GDX is searched from left to right. This follows the style of the GAMS run time projection operation:

```
option sym1<sym2, sym1<=sym2;
```

Here is an example how to load. The network is defined by the capacity parameter `cap` contained in a GDX container `net.gdx`: 
The following code loads the entire node set $n$ of the network as well as the nodes with outgoing ($out$) and incoming ($in$) arcs and the capacity $c$.

```gams
set n nodes, out(n), in(n);
parameter c(n,n) capacity
$gdxin net
$loadM n<=cap n<cap
$loadDC out<cap.dim1 in<cap.dim2 c=cap
display n, out, in;
```

The listing file looks as follows:

```
---- 6 SET n  nodes
1, 2, 3, 4, 5, 6, 7, 8, 9
---- 6 SET out Domain loaded from cap position 1
1, 2, 3, 4
---- 6 SET in  Domain loaded from cap position 2
4, 5, 6, 7, 8, 9
```

There is a potential issue with loading domains from parameters that have a zero value for some record. Since GAMS works with sparse data, it is sometime difficult to distinguish between a record with value zero (0) and the non-existence of a record. This is usually not a problem since we know the domain of a parameter and hence know all potential records. In case of using a parameter to define the domain this represents a source of confusion. Moreover, GDX has the capability of storing true zeros (most GDX utilities like gdxxrw have options (Squeeze=Y or N) to either write a true 0 or squeeze the 0s when writing GDX). So in case GDX has a zero record, a domain load from such a parameter will include this record. Here is an example. The spreadsheet Book1.xlsx contains the following data:

![Excel spreadsheet](image)

The GDX utility gdxxrw with the following command line:

```sh
gdxxrw Book1.gdx -o Book1.dat -m net -f Book1.xlsx
```
Reads the Excel data and produces a GDX container Book1.gdx with a one dimensional parameter \( \text{dat} \) (*) which can be viewed in the GDX browser in the GAMSIDE:

```
set i;
parameter a(i);
$gdxin Book1.gdx
$load i<dat.dim1 a=dat
display i,a;
```

This results in a listing file

```
---- 5 SET i  Domain loaded from dat position 1
  a1,  a2,  a4

---- 5 PARAMETER a
  a1 5.000,  a2 1.000

---- 9 PARAMETER a0
  a1 5.000,  a2 1.000,  a4 EPS
```

With gdxxrw parameter Squeeze=Y the listing file would look as follows:
2.1.2 Compile Phase Writing Data

Writing to a GDX file during compilation

<table>
<thead>
<tr>
<th>Directive</th>
<th>Parameter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$GDXOUT</td>
<td></td>
<td>Close the current GDX output file</td>
</tr>
<tr>
<td>$GDXOUT</td>
<td>Filename</td>
<td>Specify the GDX file for writing</td>
</tr>
<tr>
<td>$UNLOAD</td>
<td>id1 id2 ... idn</td>
<td>Write GAMS symbols id1, id2, ... idn to the GDX file</td>
</tr>
<tr>
<td>$UNLOAD</td>
<td>id1=gdxid1 id2=gdxid2</td>
<td>Write the GAMS symbol id1 to the GDX file with name gdxid1</td>
</tr>
<tr>
<td></td>
<td>(no identifiers)</td>
<td>(no identifiers) Write all symbols to the gdx file</td>
</tr>
</tbody>
</table>

Notes:

- Only one GDX file can be open at the same time.
- When writing data, an existing GDX file will be overwritten with the new data; there is no merge or append option.

2.2 Execution phase

During execution, we can read and write GDX files with the following statements:

To read data

```gams
execute_load 'filename', id1, id2=gdxid2, ..;
execute_loadaddc 'filename', id1, id2=gdxid2, ..;
```

The `execute_load` statement acts like an assignment statement, except that it does not merge the data read with the current data; it is a full replacement. The same restrictions apply as in an assignment statement: we cannot assign to a set that is used as a domain, or to a set used as a loop control. With `execute_loadaddc` any domain violation will be reported and flagged as execution error. In contrast, `execute_load` ignores all domain violations and loads only data that meets the domain restrictions. In addition to loading data for sets, parameters and variables, we can load a field of a variable into a parameter. Warning: when loading a single field, all other fields are reset to their default value.

To write data
execute_unload    'filename',id1,id2=gdxid,..;
execute_unloaddi  'filename',id1,id2=gdxid,..;
execute_unloadidx 'filename',id1,id2=gdxid,..;

The `execute_unload` statement replaces an existing file with that name; it does not add symbols to
or replace symbols in an existing GDX file. Without specifying any identifier, all sets, parameters,
variables and equations will be written to the GDX file. `execute_unloaddi` does the same as
`execute_unload`, but also writes the domains of all unloaded symbols to the same file.

The `execute_unloadidx` statement requires that each symbol written is a parameter; each
parameter must have a domain specified for each index position. These domains have the requirement
that they are formed using an integer sequence for the UELs that starts at 1 (one). The domain names
are changed to indicate the size of each domain. This information is used when reading the data back
from the GDX file using `$LoadIDX` during compilation. Using the special domain names, the UELs for the
domains can be recovered without writing the domains to the GDX file; see example below.

The GAMS option `gdxUELs` controls which UELs are registered in `filename`. With option `gdxUELs
= squeezed`; (default) only the UELs that are required by the exported symbols are registered while all
known UELs are registered if we set option `gdxUELs = full`.

Write a solution point

```plaintext
save_point = n
```

This is an option, specified on the command line, using an option statement or a model attribute, to write
the current model solution to a GDX file. The option values are:

0: do not write a point file (default)
1: write the solution to `<workdir><modelname>_p.gdx`
2: write the solution to `<workdir><modelname>_p<solvenumber>.gdx`

Read a solution

```plaintext
execute_loadpoint 'filename';
execute_loadpoint 'filename',id1,id2=gdxid2,..;
```

The `execute_loadpoint` allows you to merge solution points into any GAMS database. Loading the
data acts like an assignment statement and it merges/replaces data with current data. For variables and
equations, only level values and marginal values (.L and .M) are used. If no symbols are specified, all
symbols that match in type and dimensionality will be loaded into the GAMS database.

The gdx file that can be used is not limited to files created with `SAVE_POINT`; any gdx file can be used.

### 2.2.1 Execution Phase Writing Data

Example 4:
This example again uses the transport.gms model. After solving the model, we write the sets I and J and the variables Z and X to the GDX file:

```gams
*Example 4
Set I /.../,
    J /.../;
Variable X(I,J),
    Z;
...
Solve transport using LP minimizing Z;
Execute_Unload 'results.gdx',I,J,Z,X;
```

### 2.2.2 Indexed writing and reading of data

This example shows the use of the indexed write and read data:

```gams
Set I /1*100/,
    J /1*50/;
parameter A(I,J) /1.1=11, 1.9=19, 10.1=101/;
Execute_UnloadIDX 'data.gdx', A;
```

Viewing the file data.gdx in the gamside shows the modified domain information:

![Gamside showing data.gdx](image1)

To read from data.gdx, we use the indexed read:

```gams
Set I,J;
parameter A(I,J);
*load the data
$gdxin data.gdx
$LoadIDX A
$gdxin
*write all symbols so we can inspect in the gamside
$gdxout test.gdx
$unload
$gdxout
Execute_UnloadIDX 'data.gdx', A;
```

Viewing the file test.gdx in the gamside shows that the domains have been populated:
2.2.3 Writing a GDX file after compilation or execution

Using the gdx option in the GAMS call, will cause all sets, parameters, variables and equations to be written to the GDX file.

For example:

```
Gams transport gdx=transport
```

Or

```
Gams transport a=c gdx=transport
```

Using the gdx parameter when running the model using the GAMSIDE, the process window will show the GDX filename in blue indicating that the file can be opened using a double-click with the mouse.

3 Inspecting a GDX file

After creating a GDX file there are a few ways to look at its contents:

- The $LOAD directive without any parameters will show a listing of all symbols in the file.
- The GAMSIDE can be used to view the contents of a GDX file by opening the file as any other file. The IDE only recognizes the .gdx file extension.
- The GDXDUMP utility can list the symbols in the file and it also can write sets and parameters formatted as a GAMS data statement.
- The GDXDIFF utility can be used to compare two GDX files by creating a third GDX file containing the differences between all symbols with the same name, type and dimension.

4 GDX utilities

This section describes a number of GDX (GAMS Data Exchange) file utilities:

- **GDXXRW** Allows reading and writing of an Excel spreadsheet. This utility requires the presence of Microsoft Excel and therefore can only be used on a PC running the Windows operating system with Microsoft Excel installed.

- **GDXDUMP** Writes the contents of a GDX file as a GAMS formatted text file.

- **GDXDIFF** Compares the data of symbols with the same name, type and dimension in two GDX files.
files and writes the differences to a third GDX file.

- **GDXMERGE** Combines multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.

- **GDXRANK** Reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.

- **GDXCOPY** Copies/Converts one or more GDX files to a different format.

- **GDXRENAME** Rename the unique elements in a GDX file using the unique elements from second GDX file.

- **CSV2GDX** Convert a .CSV file (comma separated values) to a GDX file.

- **XLSTalk** allows for some simple communication with Excel.

- **INVERT** Perform a matrix inversion.

- **CHOLESKY** Matrix decomposition $A=LL^T$

- **EIGENVALUE** Calculates eigenvalues of a symmetric matrix

- **EIGENVECTOR** Calculates eigenvalues/vectors of a symmetric matrix

- **MCFILTER** Filter duplicate and dominated points from a solution set

## 5 GDXXRW

GDXXRW is a utility to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a `GDX` file, or read from a `GDX` file, and write the data to different ranges in a spreadsheet.

**Usage:**

```
gdxxrw Inputfile {Outputfile} {Options} [Symbols]
```

Parameters can also be read from a text file; the use of a file for parameters is indicated by preceding the file name with a `@` (At sign.). When reading parameters from a text file, lines starting with an asterisk (*) will be ignored and act as a comment.

Parameters can also be read from an area in a spreadsheet; see `Index` below.

Files without a full path name are assumed to be in the current directory when using a DOS command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory. The use of file names with embedded blanks is allowed as long as the file name is enclosed in double-quotes.
5.1 GDXXRW parameters and options

Describing the actions to be taken by gdxxrw requires passing a number parameters and options to the program. The ability of gdxxrw to process multiple actions in a single call makes the parameter passing and interpretation more complex.

There are four kinds of parameters:

Immediate

Immediate parameters are recognized and processed before any other actions are taken and can only be specified once. Examples are: `input=` `output=` `trace=`

Global

Global parameters are interpreted from left to right and affect every action that follows. The same parameter can be used multiple times to affect the actions that follow. Examples are: `SkipEmpty=` `EpsOut=`

Symbol

A symbol definition introduces a new action for reading or writing a symbol. Examples are `par=` `set=` `dset=`

Symbol attributes

Attributes specify additional information for the last symbol defined. `dim=` `cdim=` `merge=` `clear=` etc.

5.1.1 GDXXRW Options (Immediate)

Immediate options are command line options recognized independent of their position on the command line. These options are global and they can only be specified once.

Inputfile (Required parameter)

`FileName`: Is the first parameter on the command line
Or

`Input = FileName`

or

`I = FileName`
The file extension of the input file is required and determines the action taken by the program.

The extension '.gdx' for the input file will read data from a 'GDX' file and write data to a spreadsheet. The extension '.xls' or '.xlsx' for the input file will read a spreadsheet and write the data to a '.gdx' file. In addition to the '.xls' input file extension, the following file extensions are also valid for spreadsheet input files: '.wk1', '.wk2', '.wk3' and '.dbf'.

A file sharing conflict will arise when writing to a spreadsheet with the target file open in Excel. Either close the file in Excel before executing GDXXRW, or mark the spreadsheet as a shared workbook in Excel. To change the shared status of a workbook, use the Excel commands available under: Tools | Share Workbook.

Writing to a shared workbook can be painfully slow; simply closing the file and reopen the file after GDXXRW is finished is often a better option.

Outputfile (Optional parameter)

\[ \text{Output} = \text{FileName} \]

or

\[ O = \text{FileName} \]

When an output file is not specified, the output file will be derived from the input file by changing the file extension of the input file and removing any path information. The file type, file extension, depends on the installed version of Excel. Versions prior to Excel 2007 use the .xls file extension, later version use .xlsx. Excel 2007 can write .xls files, but in that case the output file has to be specified with an .xls file extension.
Logfile (Optional parameter)

Log = FileName
or
LogAppend = FileName

Specifies the filename of the log file. When omitted, log information will be written to standard output. When using GDXXRW in a GAMS model that is started from the GAMSIDE, the output will be written to the IDE process window. Using LogAppend will add the log information to the end of the file.

Delay after opening a spreadsheet

RWait = integer (default = 0)

Introduce a delay after opening a spreadsheet before accessing the data. This parameter can be used to work around an issue we encountered that Excel indicated it was not ready.

Check if any works needs to be done

CheckDate

When specified, no data will be written if the output file exists and the file date for the output file is more recent than the file date for the input file. Provides a simple check to update the output file only if the input file has changed.

Use RC notation to specify cells and ranges

UseRC

Specify that all cell and range references use RC notation. So instead of specifying the range Sheet1!A1:D6 one specifies Sheet1!R1C1:R6C4 When tracing is enabled ranges will be reported in RC notation. This is a global option and applies to all cell references.

Trace and debug output

Trace = integer (default = 1)

Sets the amount of information written to the log. Higher values will generate more output. Valid range is 0..3.

Maximum number of duplicate records allowed for a symbol

MaxDupeErrors = integer (default = 0)

Sets the maximum number of duplicate records that is allowed when reading a spreadsheet and writing to a gdx file. The duplicate records will be reported in the logfile, and if their number does not exceed the maximum specified using this option, the gdx file will not be deleted. This is a global option and applies to each symbol read from the spreadsheet.

Updating of cells that refer to other spreadsheets
UpdLinks = integer (default = 0)

Specifies how links in a spreadsheet should be updated. The valid range is 0..3.

0 Doesn't update any references
1 Updates external references but not remote references
2 Updates remote references but not external references
3 Updates both remote and external references

Execution of Excel Auto macros

RunMacros = integer (default = 0)

This option controls the execution of the 'Auto_open' and the 'Auto_close' macros when opening or closing a spreadsheet. Valid values are 0..3.

0 Doesn't execute any macros
1 Executes Auto_open macro
2 Executes Auto_close macro
3 Executes Auto_open and Auto_close macro

5.1.2 GDXXRW Symbols

To write data to a spreadsheet or to a GDX file, one or more symbols and their associated Excel range need to be specified. See also Excel Ranges

The general syntax for a Symbol specification is:

```
DataType = SymbolName {DataRange} {Dimensions} {SymbolOptions}
```

(Also see the Text directive below to write text to a spreadsheet)

**DataType**

Par = GAMS_Parameter

Specify a GAMS parameter to be read from a GDX file and written to spreadsheet, or to be read from a spreadsheet and written to a GDX file.

When writing to a spreadsheet, special values such as Eps, NA and Inf will be written in ASCII. When reading data from a spreadsheet, the ASCII strings will be used to write the corresponding special values to the GDX file.

Equ = GAMS_Equation

Var = GAMS_Variable

A sub-field of a variable or equation can be written to a spreadsheet and should be specified as part of the SymbolName. The fields recognized are .L (level), .M (marginal) .Lo (lower bound) .Up (upper bound) .Prior (priority) and .Scale (scale) The sub-field names are not case sensitive.

A sub-field of a variable or equation cannot be read from a spreadsheet and written to a GDX file.
Set = \texttt{GAMS\_Set} [\texttt{Values = Value}\texttt{Type}]

In GAMS we can define a set by specifying all its elements. In addition, each tuple can have an associated text. To read a set from a spreadsheet, the values option is used to indicate if there is any data, and if there is, if the data should be interpreted as associated text or as an indicator whether the tuple should be included in the set or not.

<table>
<thead>
<tr>
<th>ValueType</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>Based on the range, row and column dimensions for the set, the program decides on the value type to be used. This is the default for Values</td>
</tr>
<tr>
<td>NoData</td>
<td>There is no data range for the set; all tuples will be included</td>
</tr>
<tr>
<td>YN</td>
<td>Only those tuples will be included that have a data cell that is not empty and does not contain '0', 'N' or 'No'</td>
</tr>
<tr>
<td>String</td>
<td>Only those tuples will be included that have a data cell that is not empty. The string in the data cell will be used as the associated text for the tuple</td>
</tr>
<tr>
<td>All</td>
<td>All tuples will be included. A string in the data cell will be used as the associated text for the tuple</td>
</tr>
</tbody>
</table>

The following table summarizes which ValueType will be used if a value type was not specified:

<table>
<thead>
<tr>
<th>Range specification</th>
<th>Rdim = 0 Or Cdim = 0</th>
<th>Rdim &gt; 0 And Cdim &gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top left corner only</td>
<td>All</td>
<td>YN</td>
</tr>
<tr>
<td>A block, but the data range is empty</td>
<td>All</td>
<td>YN</td>
</tr>
<tr>
<td>A block, and there is a data range</td>
<td>All</td>
<td>YN</td>
</tr>
</tbody>
</table>

DSet = \texttt{GAMS\_Set}

A domain set is used to read the domain of a set from a spreadsheet row or column. Either the row or the column dimension (Rdim or Cdim) should be set to '1' to specify a row or column for the set, resulting in a one-dimensional set. Duplicate labels in the range specified do not generate an error message.

Text = "String of characters" (DataRange)

TextID = \texttt{Identifier} (DataRange)

Write the text to the cell specified in the DataRange. In addition, TextID will write the explanatory text of the Identifier in the cell to the right of the DataRange.

A Text directive can be followed by a Link=\texttt{Address} or LinkID=\texttt{identifier} directive. Using Link will create a hyperlink to an external page or to a cell in the spreadsheet. LinkID will create a hyperlink to the top left corner of the symbol specified.

The following options apply to the symbol preceding the option, and only affect that symbol:

\textit{DataRange} (Optional)
Rng = Excel Range

The Excel Range for the data for the symbol. Note that an empty range is equivalent to the first cell of the first sheet.

Dimensions (Optional)

Dim = integer

The total dimension for the symbol

Cdim = Integer

Column dimension: the number of rows in the data range that will be used to define the labels for the columns. The first Cdim rows of the data range will be used for labels.

Rdim = Integer

Row dimension: the number of columns in the data range that will be used to define the labels for the rows. The first Rdim columns of the data range will be used for the labels.

More about dimensions:

When reading data from a GDX file and writing to a spreadsheet, the dimension of the symbol is known. When reading a spreadsheet and writing to a GDX file, the dimension is not known.

The sum of Cdim and Rdim determine the dimension of the symbol. This dimension is used when writing data to a GDX file, and is used to verify the dimension of a symbol when reading from a GDX file.

When reading a GDX file, the dimension of a symbol is known, and therefore the Cdim or Rdim parameter can be omitted. If both Cdim and Rdim are omitted, the program assumes that Cdim = 1 and Rdim = dimension – 1.

ColMerge = Integer

The number of columns that will use a previous value in that column if the cell is empty. Can only be used when reading from a spreadsheet. See ColMerge example.

Symbol Options

The options below are only valid when reading a GDX file and writing to a spreadsheet.

By default, writing data to a spreadsheet will include the row and column labels in addition to the data. The row and column labels will appear in the same order as they appear in the GDX file.

Merge

Using the ‘Merge’ option assumes that the row and column labels are in the spreadsheet already. For each value read from the GDX file, the location of the row and column labels is used to update the spreadsheet. Using the merge option will force the data to be presented in a given order using the row and column labels. Spreadsheet cells for which there is no matching row/column pair will not be changed. The matching of labels is not case sensitive.
Warning: The Merge or Clear option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row / column headings in the GDX file. Cells containing strings or numbers are not affected.

Clear
The clear option is similar as the Merge option, except that the data range will be cleared before any data is written.

IntAsText = Flag (default = Y)
Unique elements that are a proper integer can be written as text or as an integer value. The default is N, which will write the unique element as a string. Note that this impacts the sorting order and can be used when using an Excel filter on a data range.

Index = Excel Range
The Index option is used to obtain the parameters by reading from the spreadsheet directly. The parameters are read using the specified range, and treated as if they were specified directly on the command line. The first three columns of the range have a fixed interpretation: DataType, Symbol identifier and Data range. The fourth and following columns can be used for additional parameters. The column header contains the keyword when necessary, and the Cell content is used as the option value.

5.1.3 GDXXRW Options
The following options affect the symbols that follow the option. They remain in effect unless they are used again for another symbol.

Acronyms = integer (default = 0)
A non-zero value indicates that acronyms can be expected and should be processed.

If no acronym processing takes place, reading an identifier in the data section of a sheet will generate an error. Writing an acronym to a sheet will write the internal numerical representation of the acronym.

Processing acronyms:
When reading a spreadsheet, an identifier in the data section of the sheet will be interpreted as an acronym and will be written to the gdx file.

When writing to a spreadsheet, a data tuple containing an acronym will be stored using the corresponding identifier of the acronym.

CMerge = integer (default = 0)
Option indicating how to read an empty cell that is part of a merged Excel range. See CMerge example. Possible values and their interpretation are:

0 Leave the cell empty
1 Use merged value in row and column headers only
2 Use merged value in all cells
**EpsOut** = `string` (default = Eps)

String to be used when writing the value for 'Epsilon'.

**Filter** = `integer` (default = 0)

Set the Excel filter for symbols written to Excel. Using this option when reading an Excel file will result in an error. Specifying filter=1 will set an Excel filter for the row of labels labels that are closest to the data values. When there are multiple rows in a column header (CDIM > 1) we can specify a filter=2 indicating to use a row away from the data values. See filter example.

**IncRC** = `flag` (default = N)

Valid only when reading a spreadsheet.
Include Excel row and column indices when a symbol is written to the gdx file. For example, when we write a parameter P with indices I and J, without this option it will be written a P(I, J). When IncRC is enabled, the parameter will be written as P(Excel_Rows, I, Excel_Columns, J). Note that the sets Excel_Rows and Excel_Columns will be added the gdx file automatically.

**MinfOut** = `string` (default = -Inf)

String to be used when writing the value for 'Negative infinity'.

**NaIn** = `string`

String to be used when reading a value for 'Not available'; this string is used in addition to the string 'NA' and is not case sensitive.

**NameConv** = `flag`

or

**NC** = `flag`

The naming convention parameter is used to change the interpretation of an Excel range that does not contain an '!' (exclamation mark). For details see Ranges below. The default value is false.

**NaOut** = `string` (default = NA)

String to be used when writing the value for 'Not available'.

**PinfOut** = `string` (default = +Inf)

String to be used when writing the value for 'Positive infinity'.

**ResetOut**

Reset the output strings for special values to their defaults.

**Squeeze** = `flag` (default = Y)

or

**SQ** = `flag`

Writing to a spreadsheet:
The squeeze option affects the writing of sub-fields of variables and equations. A value for the field that is the default value for that type of variable or equation will not be written to the spreadsheet. For example, the default for .L (Level value) is 0.0, and therefore zero will not be written to the spreadsheet. When we set SQ=N, all values will be written to the spreadsheet.

Reading a spreadsheet:
When the squeeze option is enabled, zero values for parameters will not be written to the GDX file. When the squeeze option is disabled, zero values will be written to the GDX file. In either case, empty cells, or cells containing blanks only, will never be written to the GDX file.

\[ \text{SkipEmpty} = \text{integer} \quad (\text{default} = 1) \]

or
\[ \text{SE} = \text{integer} \]

The SkipEmpty option can be used when reading a spreadsheet, and the range is specified using the top left corner instead of a block range. The value defines the number of empty row or column cells signal the end of a block. Valid values are 0..n.

\[ \text{UndfOut} = \text{string} \quad (\text{default} = \text{Undf}) \]

String to be used when writing the value for 'Undefined'.

\[ \text{AllUELs} = \text{flag} \quad (\text{default} = Y) \]

Valid only when reading a spreadsheet. When enabled, all unique elements found in a range will be entered in the GDX file. When disabled, only those unique elements that are used in conjunction with a value will be entered in the GDX file.

\[ \text{ZeroOut} = \text{string} \quad (\text{default} = 0) \]

String to be used when writing the value for 'Zero'; by default this is '0'.

### 5.1.4 Syntax elements

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>An unsigned integer</td>
</tr>
<tr>
<td>string</td>
<td>A string of characters; a string can be quoted with single or double quotation marks.</td>
</tr>
</tbody>
</table>
| flag    | True values: 1, Y or Yes
         | False values: 0, N or No
         | (not case sensitive)                 |

### 5.1.5 GDXXRW Ranges

An Excel Range is specified using the standard Excel notation: SheetName!CellRange.

When the 'SheetName!' is omitted, the first sheet will be used. A CellRange is specified by using the TopLeft:BottomRight cell notation like A1:C12. When ':BottomRight' is omitted, the program will extend the range as far down and to the right as possible. (Using '..' in stead of ':' is supported.)

Excel also allows for named ranges; a named range includes a sheet name and a cell range. Before
interpreting a range parameter, the string will be used to search for a pre-defined Excel range with that name.

When writing to a spreadsheet and a sheet name has been specified that does not exist, a new sheet will be added to the workbook with that name. Reading a spreadsheet and using an unknown range or sheet name will result in an error.

The following table summarizes all possible input combinations and their interpretation:

<table>
<thead>
<tr>
<th>Input</th>
<th>Sheet used</th>
<th>Cell(s) used</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>First sheet</td>
<td>A1</td>
</tr>
<tr>
<td>Name</td>
<td>First sheet</td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>A1</td>
</tr>
<tr>
<td>Name!</td>
<td>Name</td>
<td>A1</td>
</tr>
<tr>
<td>Name!Name</td>
<td>First sheet</td>
<td>Name</td>
</tr>
<tr>
<td>Name1!Name2</td>
<td>Name1</td>
<td>Name2</td>
</tr>
</tbody>
</table>

5.2 GDXXRW Warning

Warning: When executing gdxxrw.exe twice and redirecting output to the same log file may result in a fatal error.

For example:

```
Gdxxrw step1 parameters > logfile
Gdxxrw step2 parameters > logfile
```

The execution of step2 may fail, because Excel will close the logfile in step1 in a delayed fashion, but return control to gdxxrw.exe immediately. Using the 'Log' or 'LogAppend' parameter will avoid this problem.

5.3 GDXXRW examples

- Read spreadsheet: Example 5
- Read spreadsheet: Example 6
- Read spreadsheet: Example 7
- Read spreadsheet: Example 8
- Read spreadsheet: Example 9
- Read spreadsheet: Example 10
- Write to spreadsheet: Example 11
- Write spreadsheet: Example 12
- Read / Write spreadsheet: Example 13
- Read / Write spreadsheet: Example 14
5.3.1  Read spreadsheet: Example 5

Assuming we want to read parameter Data1 from the file test1.xls and write the data to test1.gdx. The sheet name in a range can be omitted when it refers to the first sheet.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>i1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>i2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

GDXXRW test1.xls par=Data1 rng=A1:D3 Cdim=1 Rdim=1

5.3.2  Read spreadsheet: Example 6

The same data as in the previous example, but organized differently. We use the Dset option to read set I (in column A) and set A (in column B).

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>i1</td>
<td>a1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>i1</td>
<td>a2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>i1</td>
<td>a3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>i2</td>
<td>a1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>i2</td>
<td>a2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>i2</td>
<td>a3</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

GDXXRW test1.xls par=Data2 rng=EX2!A1 RDim=2 Dset=I rng=EX2!A1 Rdim=1 Dset=A rng=EX2!B1 Rdim=1

When using a few symbols, the command line can become too long to be practical. In such case, use a text file to hold the parameters. A parameter file can contain multiple lines to increase readability and a line starting with a '*' will be ignored.

*file example6.txt
par =Data2 rng=EX2!A1 RDim=2
Dset=I rng=EX2!A1 Rdim=1
Dset=A rng=EX2!B1 Rdim=1
GDXXRW test1.xls @example6.txt

Note:
A parameter file can also be written during the execution of a GAMS model using the GAMS PUT facility.
5.3.3 Read spreadsheet: Example 7

This example illustrates how a four dimensional parameter can be specified:

```
1    q1  q1  q2  q2
2    r1  r2  r1  r2
3  a1  b1  1   2   3   4
4  a1  b2  5   6   7   8
5  a2  b1  9   10  11  12
6  a2  b2 13  14  15  16
```

When we specify the range as a block, an empty row or column will be ignored. When we specify the top left cell only, the SkipEmpty option can be used to ignore one or more empty rows or columns. When we specify SkipEmpty=0, and cells A7, B7, G1 and G2 are empty, the range can be specified with a top left cell only:

```
GDXXRW test1.xls par=Data3 rng=EX3!A1:F6 Rdim=2 Cdim=2
```

5.3.4 Read spreadsheet: Example 8

Special values can be read and written; the division by zero error in the spreadsheet will be written as 'Undefined'.

```
A   B   C   D   E   F
1  v1  v2  v3  v4  v5  v6
2 Eps NA  Eps  +Inf -Inf #DIV/0!
```

```
GDXXRW test1.xls par=Data4 rng=EX4!A1:F2 Cdim=1
```

5.3.5 Read spreadsheet: Example 9

Example of reading a set; the result will only contain the element 's1'.

```
```
5.3.6  Read spreadsheet: Example 10

The Index option is used to read a number of parameters and sets based on information stored in the spreadsheet itself. The first row of the range is used for column headings indicating additional parameters.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>Dim</td>
<td></td>
<td>Rdim</td>
</tr>
<tr>
<td>2</td>
<td>par</td>
<td>D1</td>
<td>Ex1</td>
<td>a1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>par</td>
<td>D2</td>
<td>Ex2</td>
<td>a1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>par</td>
<td>D2</td>
<td>Ex3</td>
<td>a1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>set</td>
<td>S1</td>
<td>Ex5</td>
<td>a1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>set</td>
<td>S2</td>
<td>Ex5</td>
<td>a1:a6</td>
<td>1</td>
</tr>
</tbody>
</table>

5.3.7  Write to spreadsheet: Example 11

First, we create a GDX file using the GDX parameter in the GAMS call:

```gams
*file makedata.gms
set i /i1*i4/
  j /j1*j4/
  k /k1*k4/;
parameter v(i,j,k);
v(i,j,k)$ (uniform(0,1) < 0.30) = uniform(0,1);
```

When we run this GAMS model, the file test2.gdx will be created at the end of the run.

```gams
GAMS makedata gdx=test2
```

Using the file test2.gdx, we can write to a spreadsheet:
Write parameter V to the first cell in the first sheet; because we only specify the top left corner of the sheet, the complete sheet can be used to store the data. We do not specify the row and column dimension, so they will default to \( rdim=2 \) and \( cdim=1 \). (See dimensions [1])

Before executing this example, open the Excel file (test2.xls) and use the Excel Tools menu to make this a shared notebook. After writing to the spreadsheet, use the Excel "File Save" command to verify the changes made.

```
GDXXRW test2.gdx par=V rng=a1
```

The steps above can be combined in a single GAMS model using the Execute_Unload and Execute statements as follows:

```
set i /i1*i4/
  j /j1*j4/
  k /k1*k4/;
parameter v(i,j,k);
v(i,j,k)$uniform(0,1) < 0.30 = uniform(0,1);
Execute_Unload "test2.gdx",I,J,K,V;
Execute 'GDXXRW.EXE test2.gdx par=V rng=a1';
```

### 5.3.8 Write spreadsheet: Example 12

The second sheet of this spreadsheet contains a number of labels to illustrate the use of the merge option. Note that the values written are no longer in the same cells because they have been matched with the column and row labels. Cells that were not changed by the merge option still contain 'xxx'.

```
GDXXRW test2.gdx par=V rng=sheet2!a1 merge
```

In the previous example, the cells that were not changed still contained 'xxx'. We can clear the data range before a merge by using the 'Clear' option.

```
GDXXRW test2.gdx par=V rng=sheet2!a1 clear
```

### 5.3.9 Read / Write spreadsheet: Example 13

In the following example, we read data from a spreadsheet and save the data in a GDX file. Using the $GDXIN and $LOAD GAMS directives, we read data from the GDX file into GAMS. The GAMS program modifies the data and at the end of the run the data is saved in a new GDX file (tmp.gdx). The last step updates the spreadsheet with the modified parameter.

The data in spreadsheet test1.xls:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>i1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>i2</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.3.10 Read / Write spreadsheet: Example 14

In this example we use a modified version of the trnsport.gms model from the GAMS model library. This example illustrates:

- Compilation phase
  - Read data from a spreadsheet and create a gdx file
  - Reading sets from the gdx file
  - Using the sets as a domain for additional declarations
  - Reading additional data elements
- Execution phase
  - Solve the model
  - Write solution to a gdx file
  - Use gdx file to update spreadsheet
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Distance values</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>new-york</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>seattle</td>
<td>2.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>4</td>
<td>san-diego</td>
<td>2.5</td>
<td>1.8</td>
<td>1.4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>plant capacities</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>seattle</td>
<td>350</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>san-diego</td>
<td>600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>demand at markets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>new-york</td>
<td>325</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>chicago</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>topeka</td>
<td>275</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>freight factor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data for transport model
$onecho > taskin.txt
dset=i rng=a3:a4 rdim=1
dset=j rng=b2:d2 cdim=1
par=d rng=A2 Cdim=1 Rdim=1
par=a rng=a8 Rdim=1
par=b rng=a13 Rdim=1
par=f rng=a19 Dim=0
$offecho
$call gdxxrw.exe transportdata.xls @taskin.txt
$gdxin transportdata.gdx
sets
  i(*) canning plants
  j(*) markets;
$load i j
display i, j;
Parameters
  a(i) capacity of plant i in cases
  b(j) demand at market j in cases
  d(i,j) distance in thousands of miles
scalar f freight in dollars per case per thousand miles
$load d a b f
$gdxin
Parameter c(i,j) transport cost in thousands of dollars per case ;
c(i,j) = f * d(i,j) / 1000 ;

VARIABLES
  x(i,j) shipment quantities in cases
  z total transportation costs in thousands of dollars ;
POSITIVE VARIABLE x ;
EQUATIONS
  cost define objective function
  supply(i) observe supply limit at plant i
  demand(j) satisfy demand at market j ;
cost .. z -e- sum((i,j),c(i,j)*x(i,j));
supply(i).. sum(j,x(i,j)) =l= a(i);
demand(j).. sum(i,x(i,j)) =g= b(j);
MODEL transport /all/ ;
SOLVE transport using lp minimizing z ;
DISPLAY x.l, x.m ;
execute_unload 'transportdata.gdx', x;
execute 'gdxxrw.exe transportdata.gdx var=x.l rng=sheet2!a1' ;
5.3.11 Write spreadsheet using a filter: Example 15

The following example creates a small gdx file; the gdx file is used to write the symbol A to a spreadsheet with the filter enabled.

```
set i /i1*i2/
  j /j1*j2/
  k /k1*k2/
parameter A(i,j,k);
A(i,j,k)=uniform(0,1);
execute_unload 'test.gdx', A;
execute 'gdxxrw.exe test.gdx filter=1 par=A rdim=1 cdim=2 rng=sheet1!a1';
```

The screenshot above shows the filter in Excel. When we specify filter=2 in this example with two dimensions for the column header, the row with the filter moves away from the data range as illustrated below.

```
j1  j1  j2  j2
  k1  k2  k1  k2
i1  0.171747  0.843267  0.550375  0.301138
i2  0.292212  0.224053  0.349831  0.85627
```

5.3.12 Write spreadsheet using text and hyperlinks

The following example illustrates the use of the Text directive.

First we write some data to a gdx file and we use text directive to write text to various cells; some of the cells are hyperlinks to other locations.

```
$onecho > task.txt
text="Link to data" rng=Index!A2 linkid=A
```
Below the data for symbol A” rng=data!c2
par=A rng=data!c4

text="Back to index" rng=data!a1 link=Index!A1

text="For more information visit GAMS" rng=data!c1 link=http://
www.gams.com

$offecho

set i /i1*i9/
    j /j1*j9/;
parameter A(i,j);
A(i, j) = 10 * Ord(i) + Ord(j);
execute_unload "pv.gdx";
execute 'gdxxrw pv.gdx o=pv.xls @task.txt';

Below a screen shot showing the sheet ‘data’ created by the commands above:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Back to index</td>
<td>For more information visit GAMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Below the data for symbol A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>j1</td>
<td>j2</td>
<td>j3</td>
<td>j4</td>
<td>j5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>i1</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>i2</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>i3</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>i4</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>i5</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>i6</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>i7</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>i8</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>i9</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
</tr>
</tbody>
</table>

5.3.13 ColMerge Example

Using ColMerge to read from a spreadsheet:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>red</td>
<td>blue</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>one</td>
<td>two</td>
<td>three</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>four</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>five</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reading the above spreadsheet using the following GAMS statement:
$call gdxxrw file.xls par=A rng=B2 rdim=3 cdim=1

<table>
<thead>
<tr>
<th></th>
<th>one</th>
<th>two</th>
<th>three</th>
<th>red</th>
<th>blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;empty&gt;</td>
<td>&lt;empty&gt;</td>
<td>four</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>five</td>
<td>&lt;empty&gt;</td>
<td></td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Results in empty cells B4, B5, C4 and D5. Using the ColMerge parameter, we use the non-empty content of the previous cell in the same column as the content for the empty cell. Specifying ColMerge=2, will do this for the first two columns.

$call gdxxrw file.xls par=A rng=B2 rdim=3 cdim=1 ColMerge=2

<table>
<thead>
<tr>
<th></th>
<th>one</th>
<th>two</th>
<th>three</th>
<th>red</th>
<th>blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>four</td>
<td></td>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>five</td>
<td></td>
<td></td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

### 5.3.14 CMerge Example

Using CMerge to read merged column headers:

Spreadsheet with merged cells

Note that the label 'red' is centered over columns B, C and D and label 'green' over columns E and F. We can read this spreadsheet using the following call:

$CALL gdxxrw file.xls cmerge=1 par=A rng=A1 rdim=1 cdim=2

And display the file in the GAMSIDE:
6 GDXDUMP

The program gdxdump will write scalars, sets and parameters (tables) to standard output formatted as a GAMS program with data statements. To write to a file, use the output redirection provided by the operating system.

Usage

gdxdump filename [options]

Options

-v or -version
  Write version information only and terminate; all other options will be ignored.

Symb = identifier
  Selects a single identifier to be written.

UelTable = identifier
  Write all unique elements found in the gdx file to a set using identifier as the name for the set.

Delim = [period, comma, tab, blank]
  Selects a different delimiter to separate unique elements; period is the default.

NoHeader
  Suppress the header information when writing a single symbol; only the data for the symbol will be written, not its declaration.

Header = string
  The string supplied replaces the default header written by the program. The string can be empty if an empty header is desired; such a string can be written using two single quotes (Header = '')

NoData
  Only write the headers for the symbols; no data is written.
Symbols
Generate an alphabetical list of all symbols in the GDX file.

SymbolsAsSet
Generate a set declaration where the data represents basic information of all symbols in the GDX file.

DomainInfo
Generate an alphabetical list of all symbols in the GDX file that includes domain information. The column DomInf can have the following values:

N/A The function to get the type of domain information is not available
None No domain was specified (domain is the universe)
Relaxed The domain is relaxed, i.e. the identifiers shown do not necessarily represent one dimensional sets
Regular Regular domain; the identifiers shown are one dimensional sets

Output = filename
Write output to the file specified

Format = [normal, gamsbas, CSV]
Change the output format and the symbols written.
When using the gamsbas format, the program will not write the declarations for the symbols and only write Levels and Marginals for the variables, and Marginals for equations.
The CSV format adds column headers to the output. The CDim=y option uses the unique elements of the last dimension of the symbol as column headers for the values. If domain information is available, the column headers will be made unique if overlapping names have been used for the names of the index positions. If no domain information is available, the index names used will be of the form dim1, dim2, ...

CDim = [Y, N]
Can be used when writing a CSV file; when enabled, the unique elements of the last dimension will be used as column headers for the values.

FilterDef = [Y, N]
When enabled, default values will be filtered and not written. This option is enabled by default. For example, if the Level field (.L) of a variable is zero, the value will not be written.

Note that GDX files can also be viewed using the GAMSIDE.

6.1 GDXDUMP Example 16
After executing the model trnsport.gms using the gdx option to create the file trnsport.gdx, we use gdxdump for a listing of the symbols in the file.

```
gams trnsport gdx=trnsport
gxdump trnsport Symbols
```

The gxdump program writes the following:
Symbol Dim Type  Explanatory text
1  a  1  Par  capacity of plant i in cases
2  b  1  Par  demand at market j in cases
3  c  2  Par  transport cost in thousands of dollars per case
4  cost  0  Equ  define objective function
5  d  2  Par  distance in thousands of miles
6  demand  1  Equ  satisfy demand at market j
7  f  0  Par  freight in dollars per case per thousand miles
8  i  1  Set  canning plants
9  j  1  Set  markets
10 supply  1  Equ  observe supply limit at plant i
11 x  2  Var  shipment quantities in cases
12 z  0  Var  total transportation costs in thousands of dollars

Using the DomainInfo option:
gxdump trnsport DomainInfo

SyNr  Type  DomInf  Symbol
3  Par  Regular a(i)
4  Par  Regular b(j)
7  Par  Regular c(i, j)
10 Equ  None  cost
5  Par  Regular d(i, j)
12 Equ  Regular  demand(j)
6 Par  None  f
1 Set  None  i(*)
2 Set  None  j(*)
11 Equ  Regular  supply(i)
8 Var  Regular  x(i, j)
9 Var  None  z

Using the SymbolsAsSet option:
gxdump trnsport symbolsasset

alias (Symbol, Dim, Type, *)
set  gdxitems(Symbol,Dim,Type)  Items in the GDX file /
  "i".1."Set"  "canning plants",
  "j".1."Set"  "markets",
  "a".1."Par"  "capacity of plant i in cases",
  "b".1."Par"  "demand at market j in cases",
  "d".2."Par"  "distance in thousands of miles",
  "f".0."Par"  "freight in dollars per case per thousand miles",
  "c".2."Par"  "transport cost in thousands of dollars per case",
  "x".2."Var"  "shipment quantities in cases",
  "z".0."Var"  "total transportation costs in thousands of dollars",
  "cost".0."Equ"  "define objective function",
  "supply".1."Equ"  "observe supply limit at plant i",
  "demand".1."Equ"  "satisfy demand at market j"
/;

The default output includes the declaration header and a data statement:
GDXDIFF

The gdxdiff program compares the data of symbols with the same name, type and dimension in two GDX files and writes the differences to a third GDX file. A summary report will be written to standard output.

Usage:

gdxdiff file1 file2 [diffile] {Options}

Options:

Eps = value
Absolute difference for comparisons; see below.

RelEps = value
Relative difference for comparisons; see below.

Field = FieldName
The specified field is the only field used for deciding if a variable or equation is different. FieldName is one of the following: L, M, Up, Lo, Prior, Scale or All.

ID = Identifier
Limits the comparisons to one or more symbols; symbols not specified will be ignored. Multiple identifiers can be specified as: ID=id1 ID=id2 or as ID="id1 id2"

FldOnly
Used in combination with the Field option; all variables and equations will be written as parameters using the value of the field specified. This option cannot be used in combination with DiffOnly.

DiffOnly
Differences for Variables and Equations will be written as parameters; each parameter will have an additional index which is used to store the field name. Only fields that are different will be written. This option cannot be used in combination with FldOnly.
GDXDIFF requires two parameters, the file names of two GDX files. An optional third parameter is the name of the GDX difference file. Without the third parameter, the difference file will be ‘diffile.gdx’ in the current directory.

The use of Eps and RelEps is best described by the code fragment below.

```plaintext
AbsDiff := Abs(V1 - V2);
if  AbsDiff <= EpsAbsolute
then
    Result := true
else
    if EpsRelative > 0.0
    then
        Result := AbsDiff / (1.0 + DMin(Abs(V1), Abs(V2))) <= EpsRelative
    else
        Result := false;
```

Only symbols with the same name, type and dimension will be compared. Tuples with different values are written to the GDX difference file, and a dimension is added to describe the difference using the following labels:

'ins1' indicates that the tuple only occurs in the first file.
'ins2' indicates that the tuple only occurs in the second file.
'dif1' indicates that the tuple occurs in both files; contains the value from the first file.
'dif2' indicates that the tuple occurs in both files; contains the value from the second file.

7.1 GDXDIFF Example 17

Example 16:

In the following example, the transport model is solved twice with different capacity data. GDX files are saved for each run, and compared using gdxdiff. The shipments variable is loaded into a new variable used for a display statement. We introduce four new unique elements that are used in the difference file.

```plaintext
solve transport using lp minimizing z ;
execute_unload 'case1.gdx',a,x;
a('seattle') = 1.2 * a('seattle');
solve transport using lp minimizing z ;
execute_unload 'case2.gdx',a,x;
execute 'gdxdiff.exe case1 case2 diffile';
set difftags /dif1,dif2,ins1,ins2/;
variable xdif(i,j,difftags);
execute_load 'diffile' xdif=x;
display xdif.L;
```
8  GDXMERGE

Combines multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.

Usage:
  gdxmerge filepattern1 filepattern2 .... filepatternn
  Options:
    id=<ident1>, <ident2>...
    big=<integer>
    output = fileid

Each file pattern represents a file name or a wildcard representation using ? and *. A parameter of the form @filename, will process the commands from the text file specified.

The result of the merge will be written to a file called merged.gdx unless overwritten by the output parameter.

All symbols with matching type and dimension will be merged. By specifying the parameter id=ident1 the merge process will only be performed for the identifier(s) specified.

By default, the program reads all gdx once and stores all data in memory before writing the merged.gdx file. The big parameter is used to specify a cutoff for symbols that will be written one at a time. Each symbol that exceeds the size will be processed by reading each gdx file and only process the data for that symbol. This can lead to reading the same gdx file many times, but it allows the merging of large data sets. The formula used to calculate the cutoff is:
  Dimension * TotalNumberOfElements.
  The calculated value is doubled for variables and equations.

In addition to the symbols written, a set is added to the gdx file representing all the files processed during the merge operation. The name of the set is Merged_set_1, and is made unique by changing the number. The explanatory text for each set element contains the date and time of the gdx file processed.

Notes:
  The file 'merged.gdx', or the file specified with the output parameter, will never be used in a merge operation even if the name matches a file pattern.

  Symbols with dimension 20 cannot be merged, because the resulting symbol will have dimension 21 which exceeds the maximum dimension allowed by GAMS.

Example 17

8.1  GDXMERGE Example 18

Example 17:

In this example, we solve the transport model using different LP solvers. After each run, we write all symbols to a gdx file and merge the files into one file. The variable X is read from the merged file and displayed.
Instead of using the display statement, we can also use the GAMSIDE to view the merged.gdx file. After selecting the level field to be displayed and arranging the display:

### 9 GDXRANK

Reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.

**Usage:**

```
gdxrank inputfile outputfile
```

Each one dimensional parameter is read from the input file, sorted and the corresponding integer permutation index written to the output file using the same name for the symbol. GAMS special values such as Eps, +Inf and -Inf are recognized.

**Example**

9.1 GDXRANK Example 19

In this example we sort a parameter, create a sorted version and verify that the sort worked correctly:

```gams
set I /i1 * i6/;
parameter A(I) /i1=-Inf, i2=-Inf, i3=Eps, i4= 10, i5=30, i6=20/;
display A;
* write symbol A to gdx file
```
execute_unload "rank_in.gdx", A;

* sort symbol; permutation index will be named A also
execute 'gdxrank rank_in.gdx rank_out.gdx';

* load the permutation index
parameter AIndex(i);
execute_load "rank_out.gdx", AIndex=A;
display AIndex;

* create a sorted version
parameter ASorted(i);
    ASorted(i + (AIndex(i)- Ord(i))) = A(i);
display ASorted;

* check that the result is sorted
set C(i);
C(i)=Yes$(Ord(i) < Card(i)) and (ASorted(i) > ASorted(i+1));
display C;
Abort$(Card(C) <> 0) 'sort failed';

10  GDXRENAME

Rename the unique elements in a GDX file using the unique elements from second GDX file.

Usage:

    gdxrename data-file map-file [Options]

Options:

    -R or -Reverse
    Reverse the direction of the rename operation; see below

The gdxrename utility renames the unique elements in the data-file using a two dimensional set called map in the map-file. The renaming of the unique elements only affects the string stored for each element, and does not change the data order for the symbols in the data-file. Because no data is changing in the data-file, only the strings for the unique elements, these changes are applied to the data-file directly and no new data-file is created.

A simple example follows:

```gams
set c /r, g, b, y/;
parameter A(c);
A(c) = Ord(c);
execute_unload 'data.gdx', A;
execute 'gdxdump data.gdx';
set c /r, g, b, y/;
parameter A(c);
A(c) = Ord(c);
execute_unload 'data.gdx', A;
execute 'gdxdump data.gdx';
```
```plaintext
execute 'gdxrename data map';
scalar rc;
rc = ErrorLevel;
abort$(rc <> 0) 'GDXRENAME execution error';
execute 'gdxdump data.gdx';
```

Symbol A before the rename:
```
Parameter A(*) /
'x' 1,
'g' 2,
'b' 3,
'y' 4 /;
```

Symbol A after the rename:
```
Parameter A(*) /
'red' 1,
'green' 2,
'blue' 3,
'yellow' 4 /;
```

## 11 GDXCOPY

Convert one or more gdx files to a different format

Usage:
```
gdxcopy option infile outdir
or
gdxcopy option -Replace infile
```

The first form copies the converted files to a directory; the second form replaces the original file(s).

Instead of converting the files explicitly using the gdxcopy utility, files can also be converted by using the environment variable GDXCONVERT with values V5, V6 or V7. The values specified will be used together with the value of the environment variable GDXCOMPRESS to call GDXCOPY as soon as a gdx file is created.

The values of the environment variables can also be set using the GAMS parameters gdxconvert and gdxccompress.

Output option:

<table>
<thead>
<tr>
<th>Option</th>
<th>Target format</th>
</tr>
</thead>
<tbody>
<tr>
<td>-V5</td>
<td>Version 5</td>
</tr>
<tr>
<td>-V6U</td>
<td>Version 6 uncompressed</td>
</tr>
<tr>
<td>-V6C</td>
<td>Version 6 compressed</td>
</tr>
<tr>
<td>-V7U</td>
<td>Version 7 uncompressed</td>
</tr>
<tr>
<td>-V7C</td>
<td>Version 7 compressed</td>
</tr>
</tbody>
</table>
infile: single file or a file pattern with .gdx file extension

outdir: output directory

A current GAMS system can always handle older gdx file formats. The GDXCOPY utility provides a mechanism to convert gdx files to a prior format, so an older GAMS system can read these files.

Note: Version 7 formatted files were introduced with version 22.6 of GAMS; version 6 formatted files were introduced with version 22.3 of GAMS. Prior versions used version 5.

Some features introduced in version 7 of the gdx file format cannot be represented in older formats.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Action taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension &gt; 10</td>
<td>Symbol is ignored</td>
</tr>
<tr>
<td>Identifier longer than 31 characters</td>
<td>Truncated to 31 characters</td>
</tr>
<tr>
<td>Unique element longer than 31 characters</td>
<td>Truncated to 31 characters</td>
</tr>
<tr>
<td>Domain of a symbol</td>
<td>Domain is ignored</td>
</tr>
<tr>
<td>Aliased symbol</td>
<td>Symbol is entered as a set</td>
</tr>
<tr>
<td>Additional text for symbol</td>
<td>Additional text is ignored</td>
</tr>
</tbody>
</table>

Notes:

- The Macintosh Intel-based system (DII) which was introduced with GAMS 22.6 does not support gdx conversion into formats version 6 and version 5.
- The Solaris 10 or higher Intel-based system (SIG) which was introduced with GAMS 22.5 does not support gdx conversion into formats version 5.
- Solaris 9 or higher on Sun Sparc64 (SOX) which was introduced in GAMS 22.6 does not support gdx conversion into formats version 6 and version 5.

Example

11.1 GDXCOPY Example 20

In the example below we convert all gdx files to a compressed format for version 6.

dir
  1,219 t1.gdx
  1,740 t0.gdx
889,973 i.gdx
  1,740 pv.gdx
894,672 bytes
gdxcopy -v6c *.gdx newdir
dir newdir
  1,219 t1.gdx
  1,219 t0.gdx
203,316 i.gdx
  1,219 pv.gdx
206,973 bytes
12 CSV2GDX

CSV2GDX is a program that can read a CSV file (comma separated values) and writes to a GDX file.

Many CSV files can be read by a GAMS program directly using a table statement as illustrated in Example 21, but a number of features available in CSV2GDX make it possible to read a CVS file where the table statement cannot be used.

Sample call:

```
CSV2GDX data.csv ID=ID1 UseHeader=y Index=(1,2,6) Values=(4,5)
```

Read the file data.csv and write the file data.gdx. The csv file has a header row, columns one, two and six are to be used for the index. The values can be found in the columns four and five with the labels on the first row of columns four and five used for the last index.

**Usage**

CSV2GDX *filename* *parameters*

*filename*

The input file; the .csv file extension is assumed when no extension has been specified. Reading of a compressed input file with an optional password is supported. The gzip program in the gbin sub-directory can be used to compress a file.

Parameters can also be read from a text file; the use of a file for parameters is indicated by preceding the file name with a @ (At sign.). When reading parameters from a text file, lines starting with an asterisk (*) will be ignored and act as a comment.

**Parameters**

*ID*=<id>

Identifier for the symbol in the gdx file. Additional symbols, *dim1, dim2, ...* will be added to the gdx file, representing the domain sets for the symbol *id*.

*Output*=<filename>*

Optional output filename. If no output file is specified, the program will use the input file name and change the file extension to .gdx. If a path is not specified, the output file will be created in the current directory.

*Trace*=<integer> *(Default=1)*

Sets the amount of information written to the log. Higher values will generate more output. Valid range is 0..3.

*Index*=<list of columns>*

Identify columns to get UELS from. The columns are represented as a list of integers separated by commas. For example Index=(1,2,3,4) ; in this case the notation *(1..4)* is allowed. The order in which the columns are specified is used to specify the sequence for the unique elements in the key to store the data.

*Value*=<integer>*

Specify the column to get the value from. See also DecimalSep below.
Values=<list of columns>

Specify header columns to use for UEL/Value. When using a list of columns for the values, the
field in the first line of the column is used as the unique element to store the value in that
column. See also UseHeader and AutoCol below.

UseHeader=<boolean>

Indicate if the first row is a header row. The fields in the header row are used as unique elements
when the Values option is used. A header row is not needed or can be ignored when using the
AutoCol parameter.

StoreZero=<boolean>

Indicate if zero values are ignored or written as EPS; an empty field is always ignored.

ColCount=<integer>

Number of columns in the input file. This parameter is required if there is no header line.

CheckDate=<boolean>

Write gdx file only if the csv file is more recent than the gdx file

AutoRow=<string>

Generate automatic UELs for each row. The AutoRow string is used as the prefix for the row
label numbers. The generated unique elements will be used in the first index position shifting
other elements to the right. Using AutoRow can be helpful when there are no labels that can be
used as unique elements but also to store entries that would be a duplicate entry without a
unique row label. See Example22.

AutoCol=<string>

Generate automatic UELs for each column. The AutoCol value is used as the prefix for the
column label numbers. This option overrides the use of a header line.

FieldSep=[Comma, SemiColon, Tab]

Specify a field separator. Fields are normally separated by a comma, but this parameter allows
for some additional choices.

DecimalSep=[Period, Comma]

Specify a decimal separator. The decimal is normally a period, but this parameter allows a
comma as the decimal. Special values recognized are Eps, NA, Inf, and Undef. A string that is
not recognized as a valid number will be stored as Undef.

Password=<string>

Password for an encrypted input file.

12.1 CSV2GDX Example 21

Reading the file data.csv shown below:

```
one,two,three,four,five,six
red,red,1.1,2.2,3.3,red
red,red,4.4,5.5,Eps,green
red,green,7.7,8.8,9.9,blue
blue,blue,10,0,NA,purple
```
Assuming we want to declare the parameter A as: A(color, color, color, number) we can proceed as follows:

```gams
$call csv2gdx data.csv id=A Index=(1,2,6) Values=(3..5) UseHeader=Y StoreZero=Y
set color(*), number(*);
$gdxin data.gdx
$load color=dim1
$loadm color=dim2
$loadm color=dim3
$load number=dim4
parameter A(color, color, color, number);
$load A
$gdxin
display color, number, A;
```

12.2 CSV2GDX Example 22

Consider the following input file; note the duplicate key in the first two lines:

```
red,red,1
red,red,2
red,green,3
blue,blue,4
```

The following call uses the AutoRow parameter to add a unique row label to each row. This way a GAMS program can deal with duplicate entries and prepare for better error messages.

```gams
csv2gdx example22 id=A Index=(1,2) Value=3 ColCount=3 Autorow=Row
```
13 XLSDump

This program will write all worksheets of an Excel workbook to a gdx file. Unlike GDXXRW, the program does not require that Excel is installed. Windows platforms only.

Usage:

```
XLSDump infile outfile
```

where:

<table>
<thead>
<tr>
<th>infile</th>
<th>A valid Excel workbook</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Optional. The output gdx file. If no output file is specified, the name of the input file will be used to construct the name the output file.</td>
</tr>
</tbody>
</table>

Example:

```
Original workbook test2.xlsx

Converting this workbook using xlsdump:

xlsdump test2.xlsx

Will generate the file test2.gdx. Showing this gdx file in the GAMS IDE:
```

The parameter VF:

<table>
<thead>
<tr>
<th>Entry</th>
<th>Symbol</th>
<th>Type</th>
<th>Dim</th>
<th>Nr Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>Set</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>W</td>
<td>Set</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>WS</td>
<td>Set</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>Set</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>R</td>
<td>Set</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>VF</td>
<td>Par</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>VS</td>
<td>Set</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>VU</td>
<td>Set</td>
<td>4</td>
<td>13</td>
</tr>
</tbody>
</table>

Contents of test2.gdx

Parameter VF (values)

14 XLSTalk

XLSTalk is not a GDX utility, but as the program is often used with GDXXRW, it is discussed here.

Usage:

xlstalk <option> {-V} <file> {<parameters>}

where:

<file> : Excel file name with file extension
<option> : One of the following options; see following table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
<th>Parameter</th>
<th>Return code</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A</td>
<td>Test if Excel is running</td>
<td></td>
<td>0 Excel not running</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 Excel is running</td>
</tr>
<tr>
<td>-C</td>
<td>Close file; do not save changes.</td>
<td>&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-E</td>
<td>Test if file exists</td>
<td>&lt;file&gt;</td>
<td>0 File does not exist</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 File exists</td>
</tr>
<tr>
<td>-M</td>
<td>Status of file</td>
<td>&lt;file&gt;</td>
<td>0 Not open in Excel</td>
</tr>
</tbody>
</table>

© 2014 GAMS Development Corporation
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Return Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O</td>
<td>Open file but do not reload</td>
<td>1 File is open and is not modified 2 File is open and has been modified</td>
</tr>
<tr>
<td>-Q</td>
<td>Quit Excel if no workbooks have been modified</td>
<td>0 Excel is closed 1 Excel is still running</td>
</tr>
<tr>
<td>-R</td>
<td>Run a macro</td>
<td>&lt;file&gt; &lt;macro-name&gt; {&lt;macro-param&gt;}</td>
</tr>
<tr>
<td>-S</td>
<td>Save and close the file</td>
<td>&lt;file&gt;</td>
</tr>
<tr>
<td>-V</td>
<td>Verbose mode</td>
<td></td>
</tr>
<tr>
<td>-W</td>
<td>Wait for the user to close the file</td>
<td>&lt;file&gt;</td>
</tr>
<tr>
<td>-X</td>
<td>Excel version</td>
<td>0 Excel is not installed 1 Excel is installed; version 2003 or earlier 2 Excel is installed; version 2007 or later</td>
</tr>
</tbody>
</table>

In GAMS, the return value can be obtained by using 'errorlevel'.

```
execute 'xlstalk.exe -X';
scalar x;
x = errorlevel;
display x;
```

## 15 INVERT

**INVERT**: matrix inversion

Usage:
```
invert gdxin i a gdxout inva
```

Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdxin</td>
<td>name of gdxfile with matrix</td>
</tr>
<tr>
<td>i</td>
<td>name of set used in matrix</td>
</tr>
<tr>
<td>a</td>
<td>name of 2 dimensional parameter inside gdxin</td>
</tr>
<tr>
<td>gdxout</td>
<td>name of 2 dimensional parameter inside gdxout</td>
</tr>
<tr>
<td>inva</td>
<td>name of gdxfile for results (inverse matrix)</td>
</tr>
</tbody>
</table>

Calculates the inverse of a matrix $a(i,j)$, where $i$ and $j$ are aliased sets. The matrix inva in gdxout will contain the inverse.
15.1 Invert Example 21

$ontext
Finds the inverse of a matrix through an external program

Erwin Kalvelagen, march 2005

Reference: model gauss.gms from the model library
http://www.gams.com/modlib/libhtml/ gauss.htm
$offtext

set i /i1*i3 /;
alias (i,j);
table a(i,j) 'original matrix'
  i1   i2   i3
  i1   1   2   3
  i2   1   3   4
  i3   1   4   3
;
parameter inva(i,j) 'inverse of a';
execute_unload 'a.gdx',i,a;
execute '=invert.exe a.gdx i a b.gdx inva';
execl_load 'b.gdx',inva;
display a,inva;

16 CHOLESKY

CHOLESKY: matrix decomposition A=LL^T

Usage:
cholesky gdxin i a gdxout L

Where:

<table>
<thead>
<tr>
<th>gdxin</th>
<th>name of gdxfile with matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>name of set used in matrix</td>
</tr>
<tr>
<td>a</td>
<td>name of 2 dimensional parameter inside gdxin</td>
</tr>
<tr>
<td>gdxout</td>
<td>name of gdxfile for results (factor L)</td>
</tr>
<tr>
<td>L</td>
<td>name of 2 dimensional parameter inside gdxout</td>
</tr>
</tbody>
</table>

Calculates the Choleksy decomposition A=LL^t of a symmetric positive definite matrix A=a(i,j) where i and j are aliased sets. L will contain the Cholesky factor L[i,j]
16.1 Cholesky Example 22

$ontext
Finds the cholesky decomposition A=LL' of a positive definite symmetric matrix A through an external program

Erwin Kalvelagen, may 2008

$offtext

set i /i1*i5/;
alias (i,j);
table a(i,j) 'original matrix'
  i1  i2  i3  i4  i5
i1  64  48  24  8  8
i2  48  72  42 54 36
i3  24  42  89 107 95
i4  8  54 107 210 186
i5  8  36  95 186 187;

parameter L(i,j) 'cholesky factor';
execute_unload 'a.gdx',i,a;
execute '=cholesky.exe a.gdx i a b.gdx L';
execute_load 'b.gdx',L;
display a,L;
*
* only lower triangular part of A is used
*
table a2(i,j) 'original matrix'
  i1  i2  i3  i4  i5
i1  64
i2  48  72
i3  24  42  89
i4  8  54 107 210
i5  8  36  95 186 187;

parameter L2(i,j) 'cholesky factor';
execute_unload 'a.gdx',i,a2;
execute '=cholesky.exe a.gdx i a2 b.gdx L2';
execute_load 'b.gdx',L2;
display a2,L2

17 EIGENVALUE

EIGENVALUE, calculates eigenvalues of a symmetric matrix

Usage:
eigenvalue gdxin i a gdxout ev

Where:

<table>
<thead>
<tr>
<th>gdxin</th>
<th>name of gdxfile with matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>name of set used in matrix</td>
</tr>
<tr>
<td>a</td>
<td>name of 2 dimensional parameter inside gdxin</td>
</tr>
<tr>
<td>gdxout</td>
<td>name of gdxfile for results (eigenvalues)</td>
</tr>
<tr>
<td>ev</td>
<td>name of 1 dimensional parameter inside gdxout</td>
</tr>
</tbody>
</table>

Calculates eigenvalues of symmetric matrix a(i,j) where i and j are aliased sets.

### 17.1 EigenValue Example 23

```gams
$ontext
    Eigenvalue example.

octave:1> a=[[9  1  1;  1  9  1;  1  1  9]]
a =
    9  1  1
    1  9  1
    1  1  9

octave:2> eig(a)
ans =
    8
    8
   11
$offtext

set i /i1*i3/;
alias (i,j);
table a(i,j)
    i1  i2  i3
  i1  9   1   1
  i2  1   9   1
  i3  1   1   9
;
parameter e(i) 'eigenvalues';
execute_unload 'mat.gdx',i,a;
execute '=eigenvalue.exe mat.gdx i a ev.gdx e';
execute_load 'ev.gdx',e;
display a,e;
```
18  EIGENVECTOR

EIGENVECTOR, calculates eigenvalues/vectors of a symmetric matrix

Usage:
   eigenvector gdxin i a gdxout eval evec

Where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdxin</td>
<td>name of gdxfile with matrix</td>
</tr>
<tr>
<td>i</td>
<td>name of set used in matrix</td>
</tr>
<tr>
<td>a</td>
<td>name of 2 dimensional parameter inside gdxin</td>
</tr>
<tr>
<td>gdxout</td>
<td>name of gdxfile for results (eigenvalues)</td>
</tr>
<tr>
<td>eval</td>
<td>name of 1 dimensional parameter inside gdxout</td>
</tr>
<tr>
<td>evec</td>
<td>name of 2 dimensional parameter inside gdxout</td>
</tr>
</tbody>
</table>

Calculates eigenvalues/vectors of symmetric matrix a(i,j) where i and j are aliased sets. eval will contain the eigenvalues and evec will contain the eigenvectors.

18.1 EigenVector Example 24

```octave
$ontext
   Eigenvector example.

octave:1> a = [1 2 4 7 11; 2 3 5 8 12; 4 5 6 9 13; 7 8 9 10 14; 11 12 13 14 15]
   a =
       1  2  4  7 11
       2  3  5  8 12
       4  5  6  9 13
       7  8  9 10 14
      11 12 13 14 15

octave:2> eig(a)
   ans =
       -8.464425
       -1.116317
       -0.512109
       -0.027481
       45.120332

octave:3> [e1,e2] = eig(a)
   e1 =
       0.5550905  -0.2642556   0.2892854   0.6748602   0.2879604
       0.4820641  -0.2581518   0.2196341  -0.7349311   0.3355726
       0.2865066   0.2159261  -0.8437897   0.0411896   0.3970041
      -0.0992784   0.7711236   0.3943678   0.0055409   0.4898525
      -0.6062562  -0.4714561  -0.0238286   0.0520829   0.6378888

   e2 =
```
-8.46442  0.00000  0.00000  0.00000  0.00000  
  0.00000 -1.11632  0.00000  0.00000  0.00000  
  0.00000  0.00000 -0.51211  0.00000  0.00000  
  0.00000  0.00000  0.00000 -0.02748  0.00000  
  0.00000  0.00000  0.00000  0.00000  45.12033

$offtext

set i /i1*i5/;
alias (i,j);


table a(i,j)
    i1  i2  i3  i4  i5
  i1  1   2   4   7  11
  i2  2   3   5   8  12
  i3  4   5   6   9  13
  i4  7   8   9  10  14
  i5 11  12  13  14  15

;

parameter eval(i) 'eigenvalues';
parameter evec(i,j) 'eigenvectors';

execute_unload 'mat.gdx',i,a;
execute '=eigenvector.exe mat.gdx i a ev.gdx eval evec';
execute_load 'ev.gdx',eval, evec;

display a, eval, evec;

19  MCFilter

MCFilter: Filter duplicate and dominated points from a solution set

Usage:
   mcfILTER InputFile

The input is a gdx file with the following data:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter X(point, i)</td>
<td>Points containing binary values; use EPS when all zero for a point</td>
</tr>
<tr>
<td>Parameter F(point, obj)</td>
<td>Objectives for the points X; use EPS when all zero for a point</td>
</tr>
<tr>
<td>Parameter S(obj)</td>
<td>Direction of each objective: 1=max,-1=min</td>
</tr>
</tbody>
</table>

The output will be a gdx file called InputFile_res.gdx with the same parameters but without duplicates and dominated points.

19.1  MCFilter Example

* Generate random data to test MCFilter
  set Ri /R1*R65535/
  Xi /X1*X16/
  Fi /F1*F5/;
**parameter** X(Ri, Xi) 'the binary variables'
    F(Ri, Fi) 'calculated F variables'
    S(Fi)   'sign on the F variables';

* The following points will be unique

\[
X(Ri, Xi) = \text{mod}\left(\text{floor}(Ri.\text{ord} \div \text{power}(2, Xi.\text{off})), 2\right);
\]

\[
F(Ri, Fi) = \text{uniform}(1, 2);
\]

\[
S(Fi) = -1 + 2\left(\text{uniform}(0,1) \leq 0.5\right);
\]

**execute_unload** 'testdata.gdx', X F S;

**execute** 'mcfilter testdata.gdx';

* The file testdata_res.gdx will contain the results

Below is the partial log after running the above model:

```
mcfilter v3.
Number of records   =    65535
Number of X variables =       16
Number of F variables =        5
Loading GDX data    =      156 ms
After X Filter, count   =    65535
X Duplicate filter =       16 ms
After F Filter, count   =     1070
F Dominance filter =      249 ms
Writing GDX data     =       16 ms
```

Here we see that we have 65535 points in the initial data-set. There are no duplicates, so after applying the X filter the number remains the same (65535). There are also 5 F variables, and these are processed further. After applying the X filter, we have 65535 points. The F filter removes some points, leaving us with 1070 points in the final data-set.