GAMS GDX facilities and tools

GAMS Development Corporation

6/17/2011

Table of Contents

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 32 .

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:

Directive	Parameter(s)	Description
$GDXIN		Close the current GDX input file
	Filename	Specify the GDX file to be used for reading

$LOAD $LOADDC		List all symbols in the GDX file
	`id1 id2 … idn`	Read GAMS symbols id1, id2, … idn from the GDX file
	`id1=gdxid1 id2=gdxid2`	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file
$LOAD	`id=*`	Loads all unique elements from the gdx file into set id
		Note: $LOAD simply ignores elements that are not in the domain. $LOADDC will cause a compilation error when the data read causes a domain violation.

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

2.1.1.1 Compile Phase Example 1

The `trnsport.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 trnsport.gdx
$LOAD
```

Writes the following to the listing file:

```
Content of GDX C:\XLSFUN\TRNSPORT.GDX

Number Type      Dim  Count  Name

     1 Set        1    2   i         canning plants
     2 Set        1    3   j         markets
     3 Parameter  1    2   a         capacity of plant i in cases
     4 Parameter  1    3   b         demand at market j in cases
     5 Parameter  2    6   d         distance in thousands of miles
     6 Parameter  0    1   f         freight in dollars per case per thousand miles
     7 Parameter  2    6   c         transport cost in thousands of dollars per case
     8 Variable   2    6   x         shipment quantities in cases
     9 Variable   0    1   z         total transportation costs in thousands of dollars
    10 Equation   0    1   cost      define objective function
    11 Equation   1    2   supply    observe supply limit at plant i
    12 Equation   1    3   demand    satisfy demand at market j
```

2.1.2 Compile Phase Writing Data

Writing to a GDX file during compilation

Directive	Parameter(s)	Description
$GDXOUT		Close the current GDX output file
	Filename	Specify the GDX file for writing
$UNLOAD	id1 id2 … idn	Write GAMS symbols id1, id2, … idn to the GDX file
	id1=gdxid1 id2=gdxid2	Write the GAMS symbol id1 to the GDX file with name gdxid1
		(no identifiers) Write all symbols to the gdx file

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

```
execute_load '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. 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,..;
```

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.

Write a solution point
```
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
```
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 `trnsport.gms` model. After solving the model, we write the sets `I` and `J` and the variables `Z` and `X` to the GDX file:

```
*Example 4
Set I /. . ./,
    J / . . . /;
Variable X(I,J),
```

```
        Z;
. . .
Solve trnsport using LP minimizing Z;
Execute_Unload 'results.gdx',I,J,Z,X;
```

2.2.2 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 trnsport gdx=trnsport
```

Or

```
  Gams trnsport a=c gdx=trnsport
```

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 [8] 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 [26] Writes the contents of a GDX file as a GAMS formatted text file.

- GDXDIFF [28] 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.

- GDXMERGE [29] 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 [31] 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 32 Copies/Converts one or more GDX files to a different format.
[****] 34
- XLSTalk 34 allows for some simple communication with Excel.

- INVERT 35 Perform a matrix inversion.

- CHOLESKY 36 Matrix decomposition A=LL^T

- EIGENVALUE 37 Calculates eigenvalues of a symmetric matrix

- EIGENVECTOR 38 Calculates eigenvalues/vectors of a symmetric matrix

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 14' 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 (").

Options (Immediate) 9
Options 15
Warning use in batch files 17

Note: A small utility program is available to see if Excel is installed, to close an Excel file etc. See XLSTALK 34 for more details. To read data from an Excel file without Excel installed see XLSDump 33.

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 kind 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)

Output = *FileName*
or

o = *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.

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] 17

The general syntax for a Symbol specification is:

[DataType] 12 =*SymbolName* {*[DataRange]* 13} {*[Dimensions]* 13} {*[SymbolOptions]* 14}

(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 = *GAMS_Set* [Values = *ValueType*]

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.

ValueType	Interpretation
Auto	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
NoData	There is no data range for the set; all tuples will be included

YN	Only those tuples will be included that have a data cell that is not empty and does not contain '0', 'N' or 'No'
String	All tuples will be included. The string in the data cell will be used as the associated text for the tuple

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

Range specification	Rdim = 0 Or Cdim = 0	Rdim > 0 And Cdim > 0
Top left corner only	String	YN
A block, but the data range is empty	NoData	YN
A block, and there is a data range	String	YN

DSet = *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}

Write the text to the cell specified in the DataRange.

A Text directive can be followed by a Link=*Address* or LinkID=*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:

***DataRange* (Optional)**

Rng = *Excel Range*

The Excel Range 17 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](#) 25.

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.

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[26]. 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[24].

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 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.

SkipEmpty = *integer* (default = 1)
or
SE = *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.

UndfOut = *string* (default = Undf)

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

AllUELs = *flag* (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.

ZeroOut = *string (default = 0)*

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

5.1.4   Syntax elements

integer	An unsigned integer
string	A string of characters; a string can be quoted with single or double quotation marks.
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:

Input	Sheet used	Cell(s) used	
	First sheet	A1	
!	First sheet	A1	
Name	First sheet	Name	When nc=0
Name	Name	A1	When nc=1
Name!	Name	A1	
!Name	First sheet	Name	
Name1!Name2	Name1	Name2	

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

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.

	A	B	C	D	E
1		a1	a2	a3	
2	i1	1	2	3	
3	i2	4	5	6	
4					

```
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).

	A	B	C	D
1	i1	a1	1	
2	i1	a2	2	
3	i1	a3	3	
4	i2	a1	4	
5	i2	a2	5	
6	i2	a3	6	
7				

```
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:

	A	B	C	D	E	F	G
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	
7							

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

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 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!	
3							

```
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'.

	A	B	
1			
2	s1	Y	
3	s2	N	
4	s3	No	
5	s4	0	
6	s5		
7			

```
GDXXRW test1.xls Set=SET1 values=yn rng=Ex5!A2:B6 Rdim=1
```

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.

	A	B	C	D	E	F	
1				Dim	Rdim		
2	par	D1	Ex1!a1	2			
3	par	D2	Ex2!a1		3		
4	par	D2	Ex3!a1	4	2		
5	set	S1	Ex5!a1		1		
6	set	S2	Ex5!a1:a6		1	Values=NoData	
7							

```
GDXXRW test1.xls Index=Index!a1
```

5.3.7   Write to spreadsheet: Example 11

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

```
*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 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 [13])

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:


```
$CALL GDXXRW test1.xls Set=I rng=A2:A3 Rdim=1 Set=A rng=B1:D1 Cdim=1 Par=X rng=A1:D3 Rdim=1 Cd
$GDXIN test1.gdx
Set I(*),A(*);
$LOAD I A
Parameter X(I,A);
$LOAD X
Display I,A,X;
$GDXIN
X(I,A) = - X(I,A);
Execute_Unload 'tmp.gdx',I,A,X;
Execute 'GDXXRW.EXE tmp.gdx O=test1.xls par=X rng=EX6!A1:D3 rdim=1 cdim=1';
```

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

	A	B	C	D
1	Distance values			
2		new-york	chicago	topeka
3	seattle	2.5	1.7	1.8
4	san-diego	2.5	1.8	1.4
5				
6				
7	plant capacities			
8	seattle	350		
9	san-diego	600		
10				
11				
12	demand at markets			
13	new-york	325		
14	chicago	300		
15	topeka	275		
16				
17				
18	freight factor			
19	90			

Data for trnsport 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 trnsportdata.xls @taskin.txt

$gdxin trnsportdata.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 trnasportation 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 'trnsportdata.gdx', x;
execute 'gdxxrw.exe trnsportdata.gdx var=x.l rng=sheet2!a1' ;
```

Solution written to sheet2

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';
```

	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

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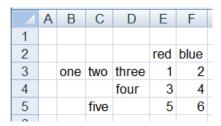
```
set i /i1*i9/
    j /j1*j9/;
parameter A(i,j);
A(i, j) = 10 * Ord(i) + Ord(j);
execute_unload "pv.gdx";
execute 'gdxxrw pv.gdx o=pv.xls @task.txt';
```

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

	A	B	C	D	E	F	G	H	I	J	K	L
1	Back to index		For more information visit GAMS									
2			Below the data for symbol A									
3												
4				j1	j2	j3	j4	j5	j6	j7	j8	j9
5			i1	11	12	13	14	15	16	17	18	19
6			i2	21	22	23	24	25	26	27	28	29
7			i3	31	32	33	34	35	36	37	38	39
8			i4	41	42	43	44	45	46	47	48	49
9			i5	51	52	53	54	55	56	57	58	59
10			i6	61	62	63	64	65	66	67	68	69
11			i7	71	72	73	74	75	76	77	78	79
12			i8	81	82	83	84	85	86	87	88	89
13			i9	91	92	93	94	95	96	97	98	99

5.3.13 ColMerge Example

Using ColMerge to read from a spreadsheet:

	A	B	C	D	E	F
1						
2					red	blue
3		one	two	three	1	2
4				four	3	4
5			five		5	6

Reading the above spreadsheet using the following GAMS statement:

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

			red	blue
one	two	three	1	2
<empty>	<empty>	four	3	4
	five	<empty>	5	6

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
```

			red	blue
one	two	three	1	2
		four	3	4
	five	\<empty\>	5	6

5.3.14 CMerge Example

Using CMerge to read merged column headers:

	A	B	C	D	E	F
1			red		green	
2		one	two	three	one	two
3	a	1	2	3	4	5
4	b	11	12	13	14	15

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:

a	red	one	1
		two	2
		three	3
	green	one	4
		two	5
b	red	one	11
		two	12
		three	13
	green	one	14
		two	15

**Symbol A
displayed in
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 for a single symbol; only the data for the symbol will be written, not its declaration.

`Symbols`
  Generate an alphabetical list of all symbols in the GDX file.

`Output =` *filename*
  Write output to the file specified

`Format = [normal, gamsbas]`
  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.

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
gdxdump trnsport
```

The gdxdump program writes the following:

```
*  GDX dump of trnsport.gdx
*  Library version    : _GAMS_GDX_V224_2002-03-19
*  File version   : _GAMS_GDX_V224_2002-03-19
*  Producer       : GAMS Rev 132  May 25, 2002 WIN.00.NA 20.6 132.000.040.
VIS P3 translator
*  Symbols        : 12
*  Unique Elements: 5
   Symbol Dim Type
 1 a        1 Par
 2 b        1 Par
 3 c        2 Par
```

```
 4 cost      0 Equ
 5 d         2 Par
 6 demand    1 Equ
 7 f         0 Par
 8 i         1 Set
 9 j         1 Set
10 supply    1 Equ
11 x         2 Var
12 z         0 Var
```

7   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.

```
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 trnsport 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.

```
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 number id 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 30

8.1 GDXMERGE Example 18

Example 17:

In this example, we solve the trnsport 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.

```
$call gamslib trnsport
$call gams trnsport lp=bdmlp gdx=bdmlp
$call gams trnsport lp=cplex gdx=cplex
$call gams trnsport lp=xpress gdx=xpress
$call gams trnsport lp=conopt gdx=conopt
$call gams trnsport lp=minos gdx=minos
$call gams trnsport lp=snopt gdx=snopt
$call gdxmerge *.gdx
variable AllX(*,*,*);
$gdxin merged.gdx
$load AllX=X
$gdxin
option AllX:5:1:2;
display AllX.L;
```

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:

	seattle		san-diego	
	new-york	chicago	new-york	topeka
bdmlp	50	300	275	275
conopt		300	325	275
cplex	50	300	275	275
minos	50	300	275	275
snopt	50	300	275	275
xpress		300	325	275

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.

9.1 GDXRANK Example 19

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

```
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   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 gdxcompress.

Output *option*:

Option	Target format
-V5	Version 5
-V6U	Version 6 uncompressed
-V6C	Version 6 compressed
-V7U	Version 7 uncompressed
-V7C	Version 7 compressed

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.

Feature	Action taken
Dimension > 10	Symbol is ignored
Identifier longer than 31 characters	Truncated to 31 characters
Unique element longer than 31 characters	Truncated to 31 characters
Domain of a symbol	Domain is ignored
Aliased symbol	Symbol is entered as a set
Additional text for symbol	Additional text is ignored

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 33

10.1 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
```

11 XLSDump

This program will write all worksheets of an Excel workbook to a gdx file. Unlike GDXXRW [8], the program does not require that Excel is installed. Windows platforms only.

Usage:
  XLSDump *infile outfile*

where:

infile	A valid Excel workbook
outfile	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.

Example:

	A	B	C	D	E	F
1						
2				a	b	c
3		i1	j1	1	2	3
4		i1	j2	4	text1	6
5		i2	j1	7	text2	9
6		i2	j3	10	11	12

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:

Entry	Symbol	Type	Dim	Nr Elem
1	S	Set	1	3
2	W	Set	1	3
3	WS	Set	2	3
4	C	Set	1	6
5	R	Set	1	6
6	VF	Par	3	10
7	VS	Set	3	13
8	VU	Set	4	13

Contents of test2.gdx

The parameter VF:

		C4	C5	C6
S1	R3	1	2	3
	R4	4		6
	R5	7		9
	R6	10	11	12

**Parameter VF
(values)**

12   **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:

Option	Action	Parameter	Return code
-A	Test if Excel is running		0 Excel not running 1 Excel is running
-C	Close file; do not save changes.	<file>	
-E	Test if file exists	<file>	0 File does not exist 1 File exists

-M	Status of file	\<file\>	0 Not open in Excel 1 File is open and is not modified 2 File is open and has been modified
-O	Open file but do not reload	\<file\>	
-Q	Quit Excel if no workbooks have been modified	\<file\>	0 Excel is closed 1 Excel is still running
-R	Run a macro	\<file\> \<macro-name\> {\<macro-param\>}	
-S	Save and close the file	\<file\>	
-V	Verbose mode		
-W	Wait for the user to close the file	\<file\>	
-X	Excel version		0 Excel is not installed 1 Excel is installed; version 2003 or earlier 2 Excel is installed; version 2007 or later

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

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

13   INVERT

INVERT: matrix inversion

Usage:
    invert gdxin i a gdxout inva

Where:

gdxin	name of gdxfile with matrix
i	name of set used in matrix
a	name of 2 dimensional parameter inside gdxin
gdxout	name of gdxfile for results (inverse matrix)
inva	name of 2 dimensional parameter inside gdxout

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.

13.1 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';
execute_load 'b.gdx',inva;

display a,inva;
```

14 CHOLESKY

CHOLESKY: matrix decomposition A=LL^T

Usage:
   cholesky gdxin i a gdxout L

Where:

gdxin	name of gdxfile with matrix
i	name of set used in matrix
a	name of 2 dimensional parameter inside gdxin
gdxout	name of gdxfile for results (factor L)
L	name of 2 dimensional parameter inside gdxout

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)

14.1 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
```

15   EIGENVALUE

EIGENVALUE, calculates eigenvalues of a symmetric matrix

Usage:
    eigenvalue gdxin i a gdxout ev

Where:

gdxin	name of gdxfile with matrix
i	name of set used in matrix
a	name of 2 dimensional parameter inside gdxin
gdxout	name of gdxfile for results (eigenvalues)
ev	name of 1 dimensional parameter inside gdxout

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

15.1 EigenValue Example 23

```
$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;
```

16 EIGENVECTOR

EIGENVECTOR, calculates eigenvalues/vectors of a symmetric matrix

Usage:
    eigenvector gdxin i a gdxout eval evec

Where:

gdxin	name of gdxfile with matrix
i	name of set used in matrix
a	name of 2 dimensional parameter inside gdxin
gdxout	name of gdxfile for results (eigenvalues)
eval	name of 1 dimensional parameter inside gdxout

evec	name of 2 dimensional parameter inside gdxout

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.

16.1   EigenVector Example 24

```
$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;
```