GDX facilities in GAMS

GDX interface to Excel
GDX text dump
GDX difference tool

Paul van der Eijk

GAMS Development Corporation

Nov 19, 2002

Table of Contents

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 gdxio.dll (gdxio.so shared objects). The interface and usage for these libraries is described in a separate document.

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.

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 set and the subsequent use of such a set as a domain.

Reading from a GDX file during compilation

Directive	Parameters	Description
$GDXIN	Filename	Specify the GDX file to be used for reading
$GDXIN		Close the current GDX input file
$LOAD		List all symbols in the GDX file
$LOAD	Id1 id2 … idn	Read GAMS symbols id1, id2, … idn from the GDX file
$LOAD	Id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file.

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

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
```

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
```

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
```

Writing to a GDX file during compilation

Directive	Parameters	Description
$GDXOUT	Filename	Specify the GDX file for writing
$GDXOUT		Close the current GDX output file
$UNLOAD	Id1 id2 … idn	Write GAMS symbols id1, id2, … idn to the GDX file
$UNLOAD	Id1=gdxid1 id2=gdxid2	Write the GAMS symbol id1 to the GDX file with name gdxid1

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.

Execution phase

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

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

and

```
execute_unload 'filename',id1,id2=gdxid,..;
```

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.

The execute_unload statement replaces an existing file with that name; it does not add symbols to or replace sysmbols in an existing GDX file.

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;
```

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.

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.

GDX utilities

This section describes three 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 and writes the differences to a third GDX file.

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 (").

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.

The program makes a distinction between immediate options and options that are processed scanning the command line from left to right. The immediate options are global and can occur only once. Other options affect how symbols are read or written; these options follow the symbol they affect.

Immediate Options:

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’ 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.

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.

Trace = *integer*

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

UpdLinks = *integer*
Specifies how links in a spreadsheet should be updated. The default value is zero, and 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

RunMacros = *integer*

This option controls the execution of the 'Auto_open' and the 'Auto_close' macros when opening or closing a spreadsheet. The default value is zero, and 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

Options

The following options affect the symbols that follow the option.

NameConv = *integer or* NC=*integer*

The naming convention parameter is used to change the interpretation of an Excel range that does not contain an '!' (exclamation mark). For details see Symbols below. Valid values are 0..1 and the default value is zero.

EpsOut = *String*

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

NaOut = *string*

String to be used when writing the value for 'Not available'; by default this is 'NA'.

MinfOut = *string*

String to be used when writing the value for 'Negative infinity'; by default this is '-Inf'.

PinfOut = *string*

String to be used when writing the value for 'Positive infinity'; by default this is '+Inf'.

UndfOut = *string*

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

ZeroOut = *string*

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

ResetOut

Reset the output strings for special values to their defaults.

Squeeze = *integer* or SQ = *integer*

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=0, all values will be written to the spreadsheet. Valid values are 0..1, and the default is one.

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* 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, and the default is zero.

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.

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 predefined 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	Cells	Comments
	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	

The general syntax for a *Symbol* is:

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

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. Cells that are empty or zero will not be written 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 no value type has been 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.

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

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 (Sheet1!A1)

Dimensions: (Optional)

Dim = *integer*

The total dimension.

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.

Symbol Options

The options below are only valid when reading a GDX file and writing to 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 = *ExcelRange*

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.

Examples to read data from a spreadsheet and create a GDX file.

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
1		a1	a2	a3
2	i1	1	2	3
3	i2	4	5	6

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

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
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 EX2!A1 Rdim=1
Dset=A 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
```

A parameter file can also be written during the execution of a GAMS model using the GAMS PUT facility.

Example 7:

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

	A	B	C	D	E	F
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

```
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
```

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
```

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	

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

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

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

Examples to read data from a GDX file and write to a spreadsheet.

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.

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';
```

Example 11:

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
```

Example 12:

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
```

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:

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

```
$CALL GDXXRW test1.xls Set=I A2:A3 Rdim=1 Set=A B1:D1 Cdim=1 Par=X
A1:D3 Rdim=1 Cdim=1
$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';
```

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:

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.

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

Example 14:

After executing trnsport as follows:

```
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
```

GDXDIFF

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*} {eps = *value*} {Field=*FieldName*}

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.

An epsilon can be specified for the comparison of two numbers.

A field can be specified to limit the comparison of variables and equations to that field. Field names follow the GAMS conventions (Lo, L, Up, M, Scale and Prio)

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.

Example 15:

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;
```

Revision History

```
Version 228
===========
GDXXRW

Added 'usage' output

Duplicate rows or columns while merging are reported

Corrected writing scalar with zero value

Squeeze option affects writing of zero for scalar and parameter

Trailing blanks are removed from spreadsheet data

Interpretation of Index is more flexible to allow for additional
parameters.

Removed SSET as a type, and added Values option for a set. Reading or
writing a SSET (string set) can now be achieved using Values=String.

SkipEmpty option can be used with Merge or Clear

When writing with Cdim=0 or Rdim=0 the data cleared is limited to the
row or column.

Added Ctrl-C and Ctrl_break handler so we can release Excel

When writing to a sheet, AutoFilterMode is disabled

GDXDIFF

Added option to specify a single field for comparison (for variables
and equations)

Added option to ignore comparison of associated text

Added RelEps option for relative comparison

Version 227
===========
GDXXRW
Test for empty row or column is now limited to the data area only. (to
the right of first column, below the first row)


Version 226
===========
GDXXRW
Added 'Rng' as prefix for range.
Allow 'Index' when writing.
Allow '..' for ':'
Output file no longer uses path of input file.

Version 225
```

```
===========
ALL COMPONENTS
Added version strings

GDXDUMP
added UelTable=
Uses DblToStr to write a double (more precision)

GDXDIFF
Corrected Eof condition
```