Using GAMS Data Exchange or GDX Files

GAMS can read or write something called a GDX file. The name GDX is an acronym for GAMS Data eXchange files. A GDX file is a platform independent, binary file that can contain information regarding sets, parameters, variables and equations. Among other usages GDX files can be used to prepare data for a GAMS model, pass results of a GAMS model into different programs, and pass results into GAMS from different programs. This feature is under current development and is likely to change as time goes on. This document covers the implementation as of September 2002. Updated information may be found in the GAMS document GDX facilities in GAMS at GAMS.

Creating a GDX file in GAMS

A GDX file can be created by GAMS in two alternative forms

A total problem summary GDX file may be created
A selected item GDX file may be created

Such files are only created on explicit user request although this may be indirect when a program like Xlexport is included which in turn creates a GDX file.

Now let's review these cases.

Command line GDX option - GDX dump of the whole problem

A composite GDX file containing all data items resident at the end of the run of a GAMS code can be created using the command line GDX parameter. The command line GDX option is invoked by adding the option GDX=filename to the GAMS command line in DOS or Unix/Linux or by including it in the command line parameter box in the IDE. The basic command line form is

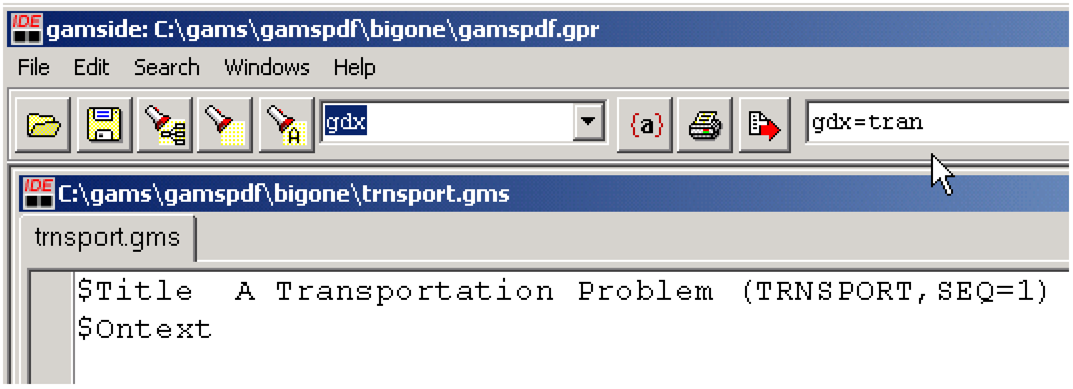
```
gams mymodelname GDX=gdxfilename
```

where

- ❖ mymodelname specifies the name of the file of GAMS instructions
- ❖ gdxfilename gives the file name and possible path where the GDX file is to be retained. When no path is specified the default directory is the current working directory or project directory in the IDE as below.

Example

An example of DOS invocation of the whole problem GDX file for the transport.gms model is given in gamsgdx.bat. When the IDE is used, the GDX file creation is invoked by an entry in the upper right hand corner of the IDE screen as illustrated below


gamside: C:\gams\gamspdf\bigone\gamspdf.gpr
File Edit Search Windows Help
gdx
gdx=tran
C:\gams\gamspdf\bigone\trnsport.gms
trnsport.gms

```
$Title  A Transportation Problem (TRNSPORT,SEQ=1)
$Ontext
```

Notes

- ❖ When this option is used the GDX file is created just at the end of the GAMS execution so the data written will contain the current values for all sets, parameters, variables and equations that are on hand at the end of the GAMS job.
- ❖ The GDX data for the variables and equations contains the levels, marginals, lower bounds, upper bounds and scales for each item.
- ❖ This yields a file that may be automatically opened in the IDE by doing a mouse click on the highlighted line in the IDE process window.

GDX files containing selected items

Selected items may be placed into a GDX file either at compile time or during execution. The syntax and effects differ so these are discussed separately.

Execution time selected item GDX file creation

An Execute_Unload command creates a GDX file containing selected problem data. The data in the GDX file are those present at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected.

The basic syntax of the statement is

```
    Execute_Unload 'filename', nameditem1,nameditem2, ... ;
```

The filename argument specifies the name of the resultant GDX file. In particular, a file with this name is created with the extension .GDX and is placed in the current working directory. This opens and closes the GDX file and does all the writing. Note the Execute_Unload command overwrites any existing file with the name filename.gdx so all writing to the file must be done in one statement.

The second part of the statement is a list of items to be placed in the GDX file and has several variants. For example, one could use multiple lines and unload several items with the command structure

```
    Execute_Unload 'filename',     nameditem1
                                   nameditem2,
                                   itemname3
                                   itemname4 ;
```

It is also possible to have different names for parameters in the GDX file and the GAMS program. In such a case, the syntax is

```
    Execute_Unload 'filename', internalname1=GDXitemname1 i2=gf2;
```

and would result in the GAMS item called internalname1 being called gdxitemname1 in the GDX file and i2 being called gf2. This syntax again can be repeated for multiple items.

<u>Example</u>

In the model gdxexectrnsport.gms we introduce the statement

```
     execute_unload 'tran2',i,j,d,f,a=sup,b=dem,x,supply;
```

The result of this is the writing of the GDX file tran2.gdx that contains the data for the sets i and j plus the parameters d, f, a and b as well as the variables x and the equations supply. In that file the a and b items have been renamed and are identified as sup and dem.

Compile time selected item GDX file creation

A group of dollar commands can be used to write a GDX file containing selected data. The data written to the GDX file will be those present when the statement is encountered during compilation. The results of calculations and solves will not be reflected. (*Note this should not ordinarily be used, it is safer to use the Execute_Unload as calculations and solves would be reflected in the result*). The only way to guarantee that the data is current is to use the execution time command or to use a save then restart a file with the dump commands within them.

The basic syntax involves a three-part sequence

```
$Gdxout filename
$Unload itemname
$Gdxout
```

The first part of the sequence is the initial $Gdxout command which also specifies the filename that the GDX file will be called. A file with this name will be placed in the current working directory with the extension .GDX. This opens the GDX file and prepares it for writing. Any existing files with the same name will be overwritten.

The second part of the sequence is one or more $Unload commands. These commands specify the items to be placed in the GDX file. A statement can specify more than one item. For example, one could unload four items with the following commands

```
$Gdxout filename
$unload itemname1
$unload itemname2
$unload itemname3
$unload itemname4
$Gdxout
```

or could accomplish the same using

```
$Gdxout filename
$unload itemname1 itemname2 itemname3 itemname4
$Gdxout
```

It is also possible to have different names for parameters in the GDX file as opposed to the names used in the GAMS program. In such a case the syntax is

```
$unload internalname1=gdxfileitemname1 i2=gf2
```

which would result in the item with internalname1 being called gdxfileitemname1 in the GDX file and i2 being called gf2.

The third part of the sequence simply consists of a $Gdxout command which closes the GDX file. Actually the statements can be intermixed with GAMS calculations solves etc. but must eventually be closed with a $Gdxout.

Example

In the model gdxtrnsport.gms we introduce the sequence

```
d(i,j)=d(i,j)*10;
$GDXout tran
$unload i j
$unload d
$unload f
$unload a=dem b=sup
$GDXout
```

The result of this is a GDX file named tran.gdx that contains the data for the sets i and j as well as the parameters d, f, a and b. Note that the a and b items have been renamed dem and sup. Also note the d items will not have been multiplied by 10 but rather take on their compile time values.

Inputting data from a GDX file into GAMS

Data in a GDX file can be read during a GAMS compile or a compile/execute sequence. GAMS can only load data from GDX files into declared items and only on an item-by-item basis. In addition GDX files are read when Xlimport is included which in turn runs a program that creates a GDX file with Excel contents and then Xlimport reads the Excel data in that GDX file.

Items may be loaded at compile time or during execution. The syntax differs depending on whether items are read at compile or execution time so these are discussed separately.

Compile time imports from GDX files

A set of dollar commands can be used to cause GAMS to read data from a GDX file at compile time. The data read from the GDX file will be the data present in it at the time that the compile job is begun

The basic syntax involves a three-part sequence

```
$Gdxin filename
$Load itemname
$Gdxin
```

The first part is an initial $Gdxin command which also specifies the filename to be used. A file with this filename and the extension .GDX is looked for in the current working directory. In turn this command opens the GDX file and prepares it for reading.

The second part of the sequence is one or more $Load commands. These commands specify the items to be read from the GDX file. Several commands may be used and each line can read more than one item. For example, one could load several items with the command structure

```
$Gdxin filename
$load itemname1
$load itemname2
$load itemname3
$load itemname4
$Gdxin
```

or could use the structure

```
$Gdxin filename
$load itemname1 itemname2 itemname3  itemname4
$Gdxin
```

It is also possible to have different names for parameters in the GDX file and the GAMS program. In such a case the syntax is

```
$load internalname1=gdxfileitemname1 i2=gf2
```

Any parameter data can be loaded as can set data defining domains and variable/equation data.

The third part of the sequence simply consists of another $Gdxin command which closes the GDX file. Actually the statements can be intermixed with GAMS calculations solves etc. but must eventually be closed with a $Gdxin.

Example

In the model gdxintrnsport.gms we introduce the sequence

```
$GDXin tran2
  Sets
       i   canning plants
       j   markets          ;
$load i j
  Parameters
       a(i)  capacity of plant i in cases
       b(j)  demand at market j in cases;
$load a=sup
$load b=dem
 Parameter d(i,j) distance in thousands of miles;
$load d
  Scalar f  freight in dollars per case;
$load f
```

```
          $GDXin
```

This loads data from the GDX file named tran2.gdx that was saved by the example gdxexectrnsport.gms.

Notes

- Items must be declared with Set, Parameter, Scalar, Variable or Equation statements before the Load appears.
- When loading items GAMS does not generate domain checking compiler errors when items are resident in GDX files for named set dependent parameters, variables, equations and sets where in the data there are references to set elements that are not present in the current file. GAMS will ignore these items and will not create errors or cause generation of any messages.
- One can import items for set positions that are not in existing sets where the set specified for that position is equivalent to the universal set (i.e. when an * is used or a terms equivalenced to the universal set or the set is a subset of the universal set).
- When the $Load is not followed by arguments this causes a listing of the GDX file contents to be generated.

Execution time GDX imports

An Execute_Load command can be used to read data from a GDX file. The data in the GDX file will be the data present in the GDX file at the time that the statement is executed and could have been updated by Execute_Unload commands during the model execution. When parameter data are loaded the Execute_Load acts like an assignment statement, except that it does not merge the data read with the current data; it is a full replacement. Sets defining domains cannot be loaded. However sets that are subsets of existing sets and do not define new elements can be loaded at execution time (Domain defining sets can be loaded can at compile time using $Load).

The basic syntax of the statement is

```
Execute_Load 'filename', nameditem1,nameditem2, ... ;
```

The filename argument specifies the name of the GDX file to read. In particular, a file with this filename with the extension .GDX will be read from the current working directory.

The second part of the statement is a list of items to be read from the GDX file. For example one could load several items with the command structure

```
Execute_Load 'filename', nameditem1
                          nameditem2,
                          itemname3
                          itemname4 ;
```

It is also possible to have different names for parameters in the GDX file and the GAMS program. In such a case the syntax is

```
    Execute_Load 'filename',internalname1=GDXitemname1 internalname2=GDXitemname2;
```

Example

In the model gdxexecintrnsport.gms we introduce the statement

```
        execute_load 'tran2',k=j,d,f,a=sup,b=dem,x,supply;
```

The result of this is that the k subset and the parameters are loaded. We also get advanced basis information when we load variables and equations.

Notes

- Items must be declared with Set, Parameter, Scalar, Variable or Equation statements before the Load appears.
- When loading data domain checking is not enforced so that when an item is resident in a GDX file for set elements not present in the current file these items are ignored and do not create errors or cause generation of any messages.

General notes on GDX files

There are several things worth noting about GDX files

- Only one GDX file can be open at the same time.
- When the GDX file to be written has the same name as an existing GDX file the existing file will be overwritten. The resultant file will only contain the new data; there is no merge or append option.
- A compile time GDX write using the $Unload will only write out data defined in the compilation at the point where the command appears. No results of any solves or calculations done within the current GAMS program will be reported with $Load. This is not true with Execute_Unload.
- An execution time GDX write using the Execute_Unload will write out data defined in the execution sequence at the point where the GDX command appears. The results of the most recent solve command and any parameter calculations occurring before the GDX write will be reported.
- Any subsequent Execute_Unload to a file written earlier will totally overwrite that file so care must be taken to write all wanted information in the last appearing Execute_Unload.

- ❖ A command line GDX write using the GDX=filename command line parameter will write out data defined at the end of the execution sequence. The results of the most recent solve and any parameter calculations will be reported.
- ❖ When loading data note that domain checking will not be enforced so that when items are resident in the GDX file for set elements not present in the current file these items will be ignored. GAMS will not generate any message to tell you items are ignored.
- ❖ Additional examples of GDX loads and unloads can be found in the library file qp1x and in the all the Performance World examples in the linlib make use of the Gdxin feature.
- ❖ Load and Unload commands provide an alternative way to load and unload a basis as opposed to GAMSBAS but in that case every variable and equation must be unloaded and loaded plus one has to be willing to stay with the same bounds and scales as they are loaded at the same time.

Identifying contents of a GDX file

Users may wish to examine the contents of a GDX file. However such files are binary and thus do not reveal information if text edited. But the GAMS system provides four ways of accomplishing this, each of which is discussed below.

Identifying contents with $Load

One can have GAMS tell you the general contents of a GDX file by using the $Load command without the name of a parameter. Namely inserting a sequence like

```
$GDXin tran2
$load
$GDXin
```

yields (gdxcontents.gms)

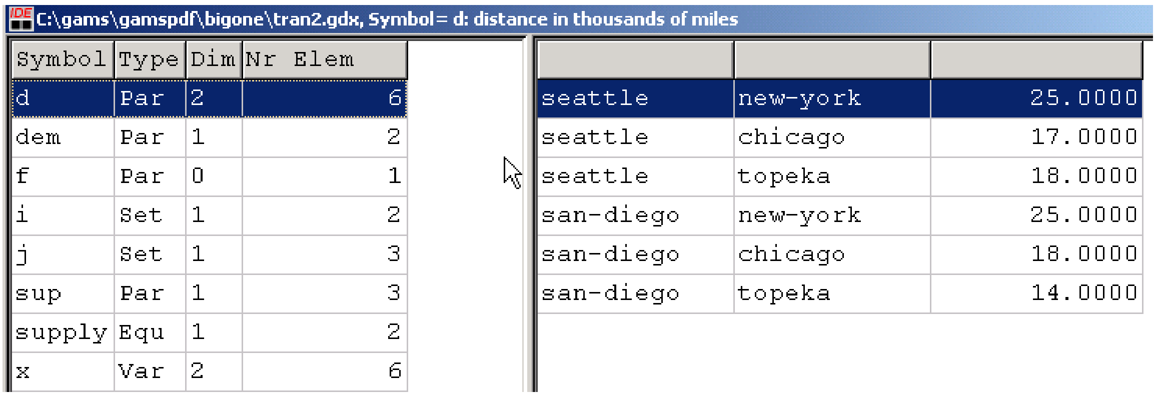
```
 Content of GDX C:\GAMS\GAMSPDF\BIGONE\TRAN2.GDX
Number Type       Dim    Count   Name

   1   Set         1       2     i        canning plants
   2   Set         1       3     j        markets
   3   Parameter   2       6     d        distance in thousands of miles
   4   Parameter   0       1     f        freight in dollars per case per thousand miles
   5   Parameter   1       2     dem      capacity of plant i in cases
   6   Parameter   1       3     sup      demand at market j in cases
   7   Variable    2       6     x        shipment quantities in cases
   8   Equation    1       2     supply   observe supply limit at plant i
```

which lists the items present by Type, Name, Number of sets the item is defined over(Dim), number of elements in the file for this item (Count).

Identifying contents with the IDE

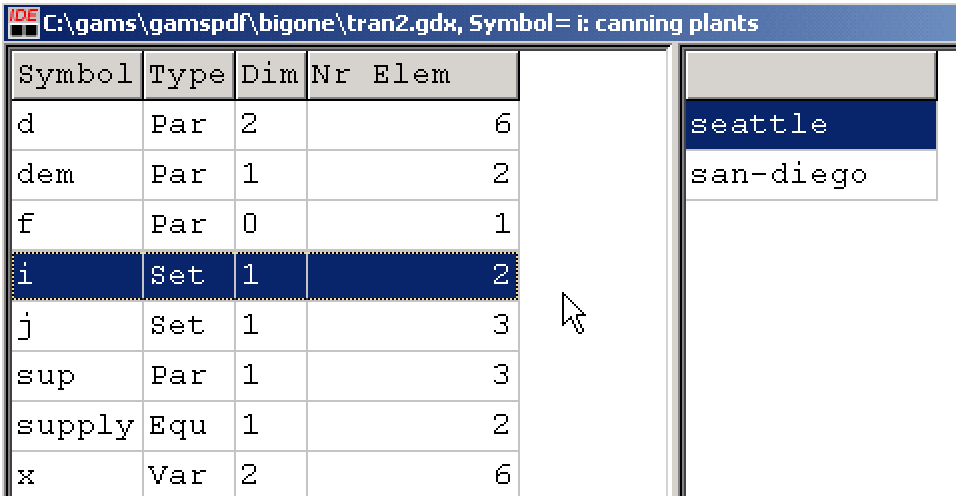
One can use the IDE to tell you the exact contents of each item in a GDX file by opening a GDX file with the Open file dialogue. Namely opening the file tran2.gdx yields the screen


C:\gams\gamspdf\bigone\tran2.gdx, Symbol= d: distance in thousands of miles

Symbol	Type	Dim	Nr Elem
d	Par	2	6
dem	Par	1	2
f	Par	0	1
i	Set	1	2
j	Set	1	3
sup	Par	1	3
supply	Equ	1	2
x	Var	2	6

seattle	new-york	25.0000
seattle	chicago	17.0000
seattle	topeka	18.0000
san-diego	new-york	25.0000
san-diego	chicago	18.0000
san-diego	topeka	14.0000

where the left hand part of the screen gives the items in the GDX field and the right hand part gives the exact data entries for the item highlighted in the left hand part. For example moving down to the set i the screens are


C:\gams\gamspdf\bigone\tran2.gdx, Symbol= i: canning plants

Symbol	Type	Dim	Nr Elem
d	Par	2	6
dem	Par	1	2
f	Par	0	1
i	Set	1	2
j	Set	1	3
sup	Par	1	3
supply	Equ	1	2
x	Var	2	6

seattle
san-diego

Also note that the columns are sortable in the left hand portion of the display. All one needs to do id to click on the gray boxes (Symbol, Type,...) with the mouse.

Identifying contents with Gdxdump

GAMS distributes a utility, Gdxdump, that will write all of the scalars, sets and parameters (tables) in a GDX file to standard output formatted as a GAMS program with data statements. It skips information for variables and equations. The syntax is

```
    Gdxdump gdxfilename
```

where the gdxfilename is the name of the GDX file to convert to GMS form. This output is created to the screen not to a file. If one wishes to dump this to a file one uses a command like

```
Gdxdump gdxfilename > filetouse.gms
```

Further details and additional options are discussed in the document GDX facilities in GAMS..

Example

For example when we use the command

```
Gdxdump gdxfilename > filetouse.gms
```

then the contents of filetouse.gms are

```
*  GDX dump of tran2.GDX
*  Library version    : _GAMS_GDX_V224_2002-03-19
*  File version    : _GAMS_GDX_V224_2002-03-19
*  Producer        : GAMS Rev 132  May 25, 2002
*  Symbols         : 8
*  Unique Elements: 5

Set i(*) canning plants/
  seattle  ,
  san-diego  /;

Set j(*) markets/
  new-york  ,
  chicago  ,
  topeka  /;

Parameter d(*,*) distance in thousands of miles/
  seattle.new-york 25 ,
  seattle.chicago 17 ,
  seattle.topeka 18 ,
  san-diego.new-york 25 ,
  san-diego.chicago 18 ,
  san-diego.topeka 14 /;

Scalar f freight in dollars per case per thousand miles/
 90 /;

Parameter dem(*) capacity of plant i in cases/
  seattle 350 ,
  san-diego 600 /;

Parameter sup(*) demand at market j in cases/
  new-york 325 ,
  chicago 300 ,
  topeka 275 /;

* skipped Variable x

* skipped Equation supply
```

where note the variable and equations are skipped at the bottom.

Identifying differences in contents with Gdxdiff

GAMS also distributes a utility that looks for differences in two GDX files creating a list of item names that differ and yet another GDX file that exactly specifies the differences.

The procedure can be run from within the IDE under the utilities choice or the DOS or Unix/Linux command line. The command line version is invoked as follows

```
Gdxdiff gdxfile1 gdxfile2 gdxdiffilename Eps=value
```

where

- gdxfile1 and gdxfile2 give the names of the GDX files to compare
- gdxdiffilename is an optional parameter naming the GDX file of differences that will be created (This will be named diffile.gdx and placed in the current directory by default.)
- Eps is an optional parameter giving the minimum difference that must be found between two numbers to signal a difference.

Further details and examples are discussed in the document GDX facilities in GAMS

Example

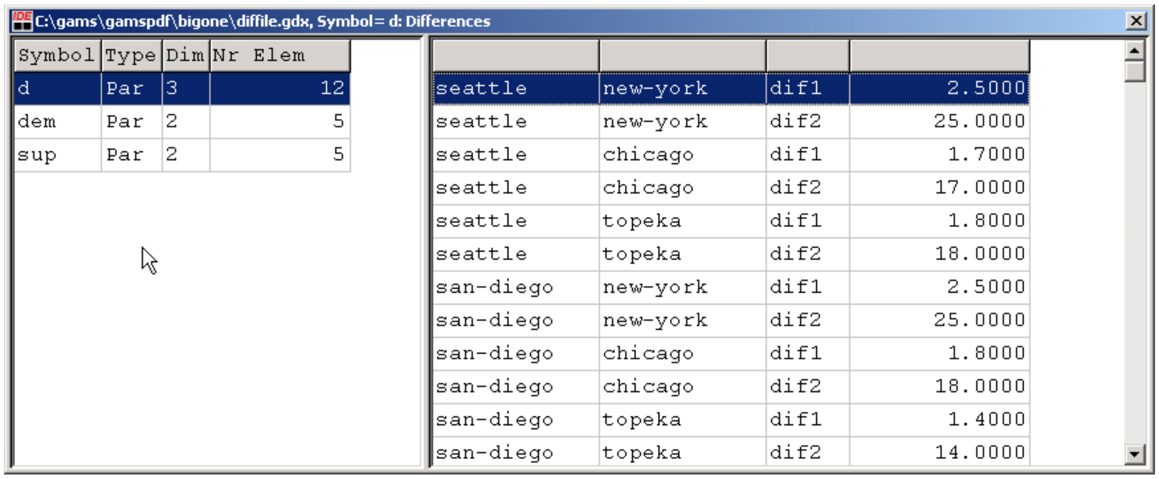
Suppose we wish to compare the GDX files tran and tran2, then we would use the command

```
Gdxdiff tran tran2
```

In turn the output to standard output (nominally the terminal screen) appears as follows

```
Summary of differences:
     d   Data is different
   dem   Keys are different
   sup   Keys are different
supply   Symbol not found in file 1
     x   Symbol not found in file 1
```

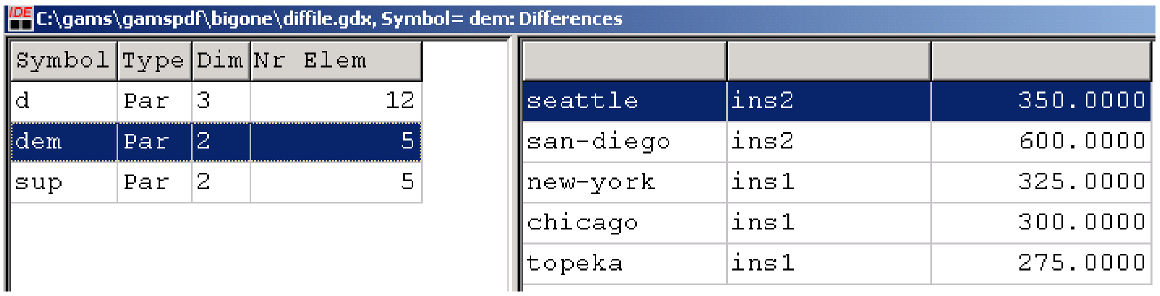
and summarizes the differences found. Simultaneously the file diffile.gdx when examined in the IDE contains the following


C:\gams\gamspdf\bigone\diffile.gdx, Symbol= d: Differences

Symbol	Type	Dim	Nr Elem
d	Par	3	12
dem	Par	2	5
sup	Par	2	5

seattle	new-york	dif1	2.5000
seattle	new-york	dif2	25.0000
seattle	chicago	dif1	1.7000
seattle	chicago	dif2	17.0000
seattle	topeka	dif1	1.8000
seattle	topeka	dif2	18.0000
san-diego	new-york	dif1	2.5000
san-diego	new-york	dif2	25.0000
san-diego	chicago	dif1	1.8000
san-diego	chicago	dif2	18.0000
san-diego	topeka	dif1	1.4000
san-diego	topeka	dif2	14.0000

which reports on the differences found in the two files. A second example for the dem parameter is


C:\gams\gamspdf\bigone\diffile.gdx, Symbol= dem: Differences

Symbol	Type	Dim	Nr Elem
d	Par	3	12
dem	Par	2	5
sup	Par	2	5

seattle	ins2	350.0000
san-diego	ins2	600.0000
new-york	ins1	325.0000
chicago	ins1	300.0000
topeka	ins1	275.0000

<u>Notes</u>

- ❖ Some new coding is introduced in the difference GDX file. Namely a new dimension is added to the parameters being compared which can contain 4 entries
  - ➢ dif1 indicates that the entry occurs in both files and shows the value found in the first file.
  - ➢ dif2 indicates that the entry occurs in both files and shows the value found in the second file.
  - ➢ ins1 indicates that the entry only occurs in the first files and shows the value found.
  - ➢ ins2 indicates that the entry only occurs in the second file and shows the value found
- ❖ Only named items with the same name, type and dimension will be compared in the diffile.gdx output. Named items that are new or are deleted will only appear in the standard output summary report.

Using GDX files to interface with other programs

The very name GDX – GAMS data exchange suggests this is the mechanism via which users will be able to exchange data with other programs. Today however this usage, while contemplated, is still under development and only exists for selected cases. In particular, there are mechanisms for spreadsheets and a couple of other programs. Let me briefly cover these.

Spreadsheets

There are currently three GDX supported pathways for data exchange to spreadsheets

- Rutherford's Xlexport, Xldump and Xlimport
- Gdxxrw
- Gdxviewer

All are discussed in the chapter Links to Other Programs Including Spreadsheets.

Other

Utilities for other types of exchanges are now under development as is a general set of procedures for reading and writing GDX files. Users needing to do such exchanges should contact GAMS Development.

Alphabetic list of features

=	Symbol to rename entries in GDX files
Dif1, dif2	Markings that indicates difference in entries in GDX files.
Domain checking	Lack of when reading GDX files
Execute_Load	Execution time GDX file element reading
Execute_Unload	Execution time GDX file creation
Gdx	GAMS data exchange file
Gdx	Creating GDX files with command line parameter
Gdx	Selected item GDX file
Gdx	Whole problem GDX file
Gdx	Viewing GDX files in the IDE
Gdx file	Creating a GDX file in GAMS
Gdxdiff	Utility to compare contents differences in two GDX files
Gdxdump	Utility to write out contents of GDX file in GAMS format
$Gdxin	Compile time GDX file naming, opening and closing
$Gdxout	Compile time GDX file naming, creation and closing
$Gdxout	Problems with compile time write to GDX

IDE	Viewing GDX files in the IDE
Ins1, ins2	Markings that indicates inserts or deletions in GDX files.
$Load	Compile time read from GDX file element identification
$Load	Listing GDX file contents
$Unload	Compile time write to GDX file element identification
$Unload	Problems with compile time write to GDX