GDX2ACCESS: A TOOL TO CONVERT GDX DATA TO MS ACCESS TABLES

ERWIN KALVELAGEN

Abstract. This document describes the GDX2ACCESS utility which allows to convert data stored in a GDX file into tables in an MS Access .MDB file.

1. Overview

GDX2ACCESS is a tool to dump the contents of a GDX file to an MS Access file (.mdb file). Every identifier gets its own table in the .MDB file. For instance when we save the results of the `trnsport` model from the model library:

```
C:\tmp>gamslib trnsport
Model trnsport.gms retrieved

C:\tmp>gams trnsport gdx=trnsport lo=2

C:\tmp>gdxdump trnsport.gdx symbols
*  GDX dump of trnsport.gdx
*  Library version: _GAMS_GDX_233_2005-03-03
*  File version   : _GAMS_GDX_233_2005-03-03
*  Producer       : GAMS Rev 142  BETA  8Mar05 WIN.00.NA 21.7 142.000.041.VIS P3PC
*  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

C:\tmp>
```

The example shows how we copy the `trnsport.gms` model from the model library, and then solve it. The option `gdx=filename` will save the complete symbol table to a GDX file. The option `lo=2` tells GAMS to save the log to a file (in this case `trnsport.log`) instead of writing it to the screen. The `gdxdump` will display the contents of the GDX file (the option `symbols` will only display the table of contents, rather than all data).

Once we have a GDX file we can use GDX2ACCESS to create an .MDB file:

```
C:\tmp>gdx2access trnsport.gdx
GDX2ACCESS Version 2.0
Renaming trnsport.MDB -> trnsport.BAK
Opening C:\tmp\trnsport.MDB with Access: 0.13 seconds
Using temp directory C:\DOCUME~1\HP_Owner\LOCALS~1\Temp\
          i.   Insert:  0.02 seconds
          j.   Insert:  0.02 seconds
          a.   Insert:  0.00 seconds
          b.   Insert:  0.02 seconds
          d.   Insert:  0.03 seconds
          f.   Insert:  0.00 seconds
          c.   Insert:  0.02 seconds
          x.   Insert:  0.02 seconds
          z.   Insert:  0.00 seconds
       cost.   Insert:  0.00 seconds
     supply.   Insert:  0.00 seconds
     demand.   Insert:  0.02 seconds
```

Date: May 25, 2005.

```
Total elapsed time: 0.47 seconds
C:\tmp>
```

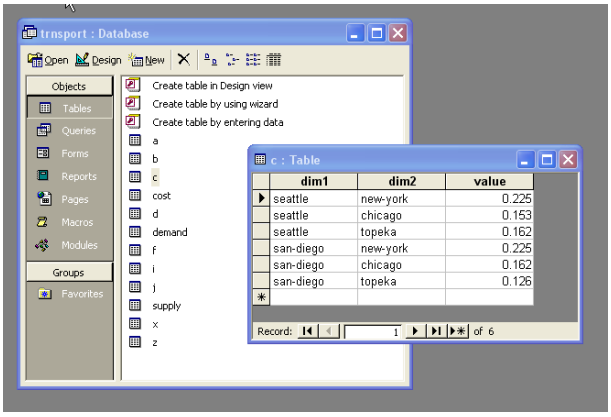

FIGURE 1. Trnsport.gdx converted to an Access database

The resulting MDB, opened with MS Access is shown in figure 1.

The complete process shown here can be automated as is shown in section 4.1. As can be seen, every identifier is stored in its own table. Index positions get a column with labels *dim1*, *dim2*, etc.

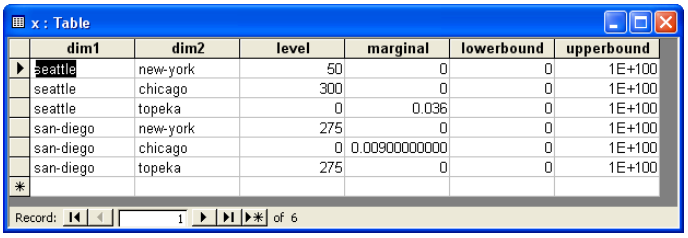

FIGURE 2. A variable stored in the database

For parameters, the value is stored in a column names *value*, while variables and equations have columns *level*, *marginal*, *lowerbound*, *upperbound*. A possible additional field (*scale* for NLP's, *priority* for MIP's, *stage* for stochastic problems) is not exported. If needed you can assign such a quantity to a parameter before writing the GDX file. For an example see figure 2.

2. Column names

For better column names than *dim1*, *dim2*, etc., we need to pass on extra information to GDX2ACCESS. GAMS can produce a *reference file* using the `RF` command line switch:

```
>gams trnsport gdx=trnsport.gdx rf=trnsport.ref
>gdx2access trnsport.gdx trnsport.ref
```

This will now create tables with names for index columns that reflect the set names used in the GAMS model.

3. Options

Options are specified in an .INI file. By default, the file `gdx2access.ini` located in the same directory as `gdx2access.exe` is consulted. If this file is not available, the program will continue using default settings.

It is also possible to tell the program to use a different .ini file. This is done by using an extra argument of the form `@inifile`. An example would be:

```
C:\TMP> gdx2access myfile.gdx @myinifile.ini
```

In this case the program will not read `gdx2access.ini` located in the same directory as `gdx2access.exe` but rather `myinifile.ini` in the current directory.

The ini file can contain two sections: `[settings]` and `[debug]`. A complete ini file with all possible settings looks like:

```
[settings]
scrdir=c:\tmp
inf=1.0e100
mininf=-1.0e100
eps=0.0
na=0.0
undf=0.0
scalartable=1

[debug]
method=5
thresholdcount=5
keepfiles=1
```

A complete description for the `[settings]` section is:

scrdir: This parameter specifies the directory where GDX2ACCESS writes temporary scratch files. By default the system uses the Windows temporary file path [1].

inf: Special values need to be mapped to numeric values so the values can be stored in a numeric field. This setting will specify the value for the GAMS `INF` quantity. The default is `1.0e100`.

mininf: This is the numeric value for `-INF`. The default is `-1.0e100`.

eps: This is the numeric value to be used for `EPS`. The default is `0.0`.

na: This is the numeric value to be used for `NA`. The default is `0.0`.

undf: This is the numeric value to be used for `UNDF`. The default is `0.0`.

scalartable: When this parameter is set to 1, GDX2ACCESS will generate separate tables to collect scalar parameters, scalar equations and scalar variables. This can reduce the number of tables created with just a single row. The names of the tables is fixed: `ScalarParameter`, `ScalarEquation` and `ScalarVariable`. By default this option is turned off.

Notes: It is currently not possible to specify `NULL` values for special values. An example of setting special values can be found in section 4.4.

A complete description for the `[debug]` section is:

method: The GDX2ACCESS program contains several algorithms to insert data into an Access database. A short description is:

method=1: Write a CSV file and use the `TransferText` action to read this into Access. This is fast, but does not always work when non-US language settings are used.

method=2: Use `recordset.add` to add records. This is slow, but does not use intermediate files.

method=3: Write a tab delimited file with a complete file specification (`schema.ini`) and use the ISAM Text driver to import the data. This is fast and should work in international settings.

method=4: For small data use `method=2` and for larger data items use `method=1`.

method=5: For small data use `method=2` and for larger data items use `method=3`. The is the default.

thresholdcount: When to change between algorithms when using `method=4` or `method=5`. The default is 5 records.

keepfiles: When set to 1, GDX2ACCESS will not delete intermediate scratch files.

4. Examples

4.1. **Model gdx2access1: import trnsport.gdx.** This example will solve the `trnsport.gms` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Access and MS Access is launched to inspect the results. This is a small example that should run very quickly.

```
$ontext

 Test of GDX2ACCESS. Dumps all symbols of
 trnsport.gms to trnsport.mdb

$offtext
execute '=gamslib trnsport';
```

```
execute '=gams trnsport lo=3 gdx=trnsport';
execute '=gdx2access trnsport.gdx';
execute '=shellExecute trnsport.mdb';
```

Notes: the equal signs in from of the external programs indicate we don't go through a shell (e.g. `command.com` or `cmd.exe`). This will improve reliability in case the external program is not found. In such a case a proper error will be triggered. Without the '=' such errors go undetected and the GAMS model will continue.

The command `ShellExecute` will launch Access to view the .MDB file, see [5].

4.2. **Model gdx2access2: import indus89.gdx.** This example will solve the `indus89.gms` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Access and MS Access is launched to inspect the results. This is a fairly large GDX file, with many identifiers, resulting in many tables in the database.

```
$ontext

 Test of GDX2ACCESS. Dumps all symbols of
 indus89.gms to indus89.mdb. This takes
 longer as there is a large number of symbols.

$offtext

execute '=gamslib indus89';
execute '=gams indus89 lo=3 gdx=indus89';
execute '=gdx2access indus89.gdx';
execute '=shellExecute indus89.mdb';
```

4.3. **Model gdx2access3: a large table.** This is an artificial example where we generate a large identifier in GAMS: a parameter with a million elements.

```
$ontext

 Test of GDX2ACCESS. Dumps a large symbol (a million elements)
 to an Access Database.

$offtext

set i /i1*i1000/;
alias (i,j);
parameter p(i,j);
p(i,j) = uniform(-100,100);
execute_unload 'test.gdx',p;

execute '=gdx2access test.gdx';
execute '=ShellExecute test.mdb';
```

4.4. **Model gdx2access4: special value mapping.** To store special values like INF, EPS, NA in a numeric field in the database, GDX2ACCESS uses a mapping. This mapping can be changed using an INI file.

```
$ontext

 Test of GDX2ACCESS.
 Check special value mapping.

$offtext

$onecho > m.ini
[settings]
inf=1
mininf=2
eps=3
na=4
undf=5
$offecho


parameter p(*) /
   i1 inf
   i2 -inf
   i3 eps
   i4 na

/;
p('i5') = 1/0;
```

```
display p;


*
* save parameter p in p.mdb, as table p
* special values are translated to default values:
*  INF -> 1.0e100
*  -INF -> -1.0e100
*  EPS,NA,UNDF -> 0
*
execute_unload "p.gdx",p;
execute '=gdx2access p.gdx';

*
* now read database p.mdb again, and get parameter as q in gdx file q.gdx.
*
execute '=mdb2gms i=p.mdb x=q.gdx q="select * from p" p=q';

*
* now load q, we should only see 1.0e100 (was INF) and -1.0e100 (was -INF)
*
parameter q(*);
execute_load "q.gdx",q;
display q;

*
* save parameter p in p.mdb, as table p, using new mapping
*  INF -> 1
*  -INF -> 2
*  EPS -> 3
*  NA -> 4
*  UNDF -> 5
*
execute '=gdx2access p.gdx @m.ini';

*
* now read database p.mdb again, and get parameter as q in gdx file q.gdx.
*
execute '=mdb2gms i=p.mdb x=q.gdx q="select * from p" p=q';

*
* now load q, we should see 1,2,3,4,5
*
parameter q2(*);
execute_load "q.gdx",q2=q;
display q2;
```

We should see as results:

```
**** Exec Error at line 25: division by zero (0)

----     26 PARAMETER p

i1 +INF,    i2 -INF,    i3  EPS,    i4   NA,    i5 UNDF


----     49 PARAMETER q

i1 1.0000E+100,    i2 -1.000E+100


----     71 PARAMETER q2

i1 1.000,    i2 2.000,    i3 3.000,    i4 4.000,    i5 5.000
```

The division by zero is used to trigger an UNDF value, as we cannot specify this in a parameter initialization. We use MDB2GMS to get the database contents back into GAMS[4].

4.5. **Model gdx2access5: renaming fields I.** The GDX file does not contain domain information. I.e. a parameter `c(i,j)` is really stored as `c(*,*)`. This can be seen if we use GDXDUMP on a parameter `c(i,j)` saved using:

```
execute_unload "c.gdx",c;
execute "=gdxdump c.gdx";
```

The GDXDUMP output will look like:

```
*  GDX dump of c.gdx
*  Library version: _GAMS_GDX_234_2005-05-02
*  File version   : _GAMS_GDX_234_2005-05-02
```

```
*  Producer       : GAMS Rev 143  ALFA  1Jun05 WIN.00.NA 22.0 143.000.041.VIS P3PC
*  Symbols        :    1
*  Unique Elements:    5

Parameter c(*,*) transport cost in thousands of dollars per case /
  seattle.new-york 0.225 ,
  seattle.chicago 0.153 ,
  seattle.topeka 0.162 ,
  san-diego.new-york 0.225 ,
  san-diego.chicago 0.162 ,
  san-diego.topeka 0.126 /;
```

As a result GDX2ACCESS will invent field names like `dim1`, `dim2`,`Value`. In some cases this may not be convenient, e.g. when more descriptive field names are required. In the following model we will show how a small script in VBscript[3, 2] can handle this task. The script will rename the fields `dim1`, `dim2`,`Value` in table `c` to `i`, `j`, and `transportcost`.

```
$ontext

   GDX2ACCESS example
   Rename fields through VBScript

$offtext
sets
  i "canning plants" / seattle, san-diego /
  j "markets"        / new-york, chicago, topeka /
;

parameters
   a(i)  "capacity of plant i in cases" /
            seattle     350
            san-diego   600
           /
   b(j) "demand at market j in cases"  /
            new-york    325
            chicago     300
            topeka      275
          /
;

table d(i,j)  "distance in thousands of miles"
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4
;

scalar f  "freight in dollars per case per thousand miles"  /90/ ;

parameter c(i,j)  "transport cost in thousands of dollars per case";
c(i,j) = f * d(i,j) / 1000 ;

*
* export to gdx file.
* The domains i,j are lost: gdx only stores c(*,*)
*
execute_unload "c.gdx",c;

*
* move to access database
* column names are dim1,dim2
*
execute "=gdx2access c.gdx";

*
* rename columns
*
execute "=cscript access.vbs";

*
* view results
*
execute "=shellexecute c.mdb";


$onecho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"

' Office 2000 DAO version
' Change to local situation.
```

```
Set oDAO = CreateObject("DAO.DBEngine.36")
Wscript.Echo "DAO version : " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.mdb")
Wscript.Echo "Opened : " & oDB.name

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close

Wscript.Echo "Done"
$offecho
```

4.6. **Model gdx2access6: renaming fields II.** The VBscript example from the previous example can be more robust by not creating the DAO (Data Access Objects) 3.6 object directly, but rather let MS Access decide which DAO version to use. This version would look like:

```
$ontext

   GDX2ACCESS example
   Rename fields through VBScript

$offtext

sets
  i "canning plants" / seattle, san-diego /
  j "markets"        / new-york, chicago, topeka /
;

parameters
   a(i)  "capacity of plant i in cases" /
            seattle     350
            san-diego   600
          /
   b(j) "demand at market j in cases"  /
            new-york    325
            chicago     300
            topeka      275
         /
;

table d(i,j)  "distance in thousands of miles"
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4
;

scalar f  "freight in dollars per case per thousand miles"  /90/ ;

parameter c(i,j)  "transport cost in thousands of dollars per case";
c(i,j) = f * d(i,j) / 1000 ;

*
* export to gdx file.
* The domains i,j are lost: gdx only stores c(*,*)
*
execute_unload "c.gdx",c;

*
* move to access database
* column names are dim1,dim2
*
execute "=gdx2access c.gdx";

*
* rename columns
*
execute "=cscript access.vbs";

*
* view results
*
execute "=shellexecute c.mdb";
```

```
$onecho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"

set oa = CreateObject("Access.Application")
set oDAO = oa.DBEngine
Wscript.Echo "DAO Version: " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.mdb")
Wscript.Echo "Opened : " & oDB.name

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close

Wscript.Echo "Done"
$offecho
```

4.7. **Model gdx2access7: using a reference file.** This example shows how to use a reference file to pass on domain information to GDX2ACCESS.

```
$ontext

 Test of GDX2ACCESS. Dumps all symbols of
 trnsport.gms to trnsport.mdb. This
 version uses a ref file to provide
 domain information.

$offtext
execute '=gamslib trnsport';
execute '=gams trnsport lo=3 gdx=trnsport rf=trnsport';
execute '=gdx2access trnsport.gdx trnsport.ref';
execute '=shellExecute trnsport.mdb';
```

4.8. **Model gdx2access8: creating a reference file.** The organization of a *reference file* is rather simple. This allows us to mimic the format of this file, excluding information that is not used, so that we can force GDX2ACCESS to use better column names.

```
$ontext

 Test of GDX2ACCESS. Generate custom reference file.

 Format
    line 0 : 0 nr_of_symbols
    line 1... : symno symbolname symboltype string-ignored dimension int-ignored domain(1)..domain(dim)

symboltypes:
   2 : set
   4 : parameter
   5 : variable
   6 : equation

$offtext


sets
   i /i1*i4/
   j /j1*j5/
   k /k1*k6/
;
parameter p(i,j,k);
p(i,j,k) = uniform(0,1);
execute_unload "p.gdx",p;

$onecho > p.ref
0 4
1 i 2 - 0 -1
2 j 2 - 0 -1
3 k 2 - 0 -1
```

```
4 p 4 - 3 -1 1 2 3
$offecho

execute '=gdx2access p.gdx p.ref';
execute '=shellExecute p.mdb';
```

References

1. Microsoft Corp., *GetTempPath*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/gettemppath.asp, March 2005.
2. ———, *VBScript Language Reference*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbscripttoc.asp, 2005.
3. ———, *VBScript User's Guide*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vbstutor.asp, 2005.
4. Erwin Kalvelagen, *MDB2GMS: A Tool for Importing Ms Access Database Tables into GAMS*, http://www.gams.com/~erwin/interface/mdb2gms.pdf, March 2004.
5. ———, *ShellExecute: A Tool for Launching External Programs*, http://www.gams.com/~erwin/shellexecute.pdf, March 2004.

GAMS Development Corp.
E-mail address: erwin@gams.com