1 Contents

MDB2GMS

2 Overview

MDB2GMS is a tool to convert data from an *Microsoft Access database* into GAMS readable format. The source is an MS Access database file (*.MDB or .ACCDB) and the target is a *GAMS Include File* or a GAMS GDX File.

When running the executable MDB2GMS.EXE without command line parameters the tool will run interactively with a built-in GUI interface. Alternatively MDB2GMS can be run in batch mode which is useful when calling it directly from a GAMS model using the *$call* command.

3 Requirements

MDB2GMS runs only on PC's running Windows (95/98/NT/XP) and with MS Access installed. MS Access comes with certain versions of MS Office, but some Office versions will not include Access. The actual retrieval of the database records is performed by **DAO** or *Data Access Objects*, an object layer for accessing the database. The actual database is the **Jet** engine, which performs the queries and retrieves the data.

To use this tool effectively you will need to have a working knowledge of **SQL** in order to formulate proper database queries.

4 Converting database tables to GAMS data

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters are **single valued** indicating written as a table GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION can be used to generate the correct GAMS file.

Besides parameters it is also possible to generate **set** data. The following tables summarizes some of the possibilities:

Single valued table	`parameter d(i,j) /`
	`$include data.inc`
	`/;`
		SELECT city1,city2,distance
		FROM distances

Multi valued table
 	
Use two separate parameters and queries or a parameter with an extra index position and a UNION select.
	`parameter sales(year,loc,prd) /`
	`$include sales.inc`
	`/;`
	`parameter profit(year,loc,prd) /`
	`$include profit.inc`
	`/;`
	
	or
	
	`set typ /sales,profit/`
	`parameter data(year,loc,prd, typ) /`
	`$include data.inc`
	`/;`
		SELECT year,loc,prod,sales
		FROM data
		
		SELECT year,loc,prod,profit
		FROM data
		
		SELECT year,loc,prod,'sales',sales
		FROM data
		UNION
		SELECT year,loc,prod,'profit',profit
		FROM data

Single dimension set
 	
Make sure elements are unique
	`set i /`
	`$include set.inc`
	`/;`
		SELECT distinct(indexcolumn)
		FROM datatable

Multi dimensional set
 	
Add dummy value field to make sure the elements are separated by a dot.
	`set ij(i,j) /`
	`$include ij.inc`
	`/;`
		SELECT indx1,indx2," "
		FROM datatable

There are no special requirements on the data types used in the database. The data are converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

5 Example: single-valued tables

Consider the simple example table **Distances**:

City1	City2	Distance
SEATTLE	NEW-YORK	2.5
SAN-DIEGO	NEW-YORK	2.5
SEATTLE	CHICAGO	1.7
SAN-DIEGO	CHICAGO	1.8
SEATTLE	TOPEKA	1.8
SAN-DIEGO	TOPEKA	1.4

with the query:

```
SELECT City1,City2,Distance
FROM Distances
```

This can be represented in GAMS as:

```
set i /seattle, san-diego/;
set j /new-york, chicago, topeka/;
parameter dist(i,j) 'distances' /
$include distances.inc
/;
```

where the include file **distances.inc** has been generated using the above query. This file can look like:

```
* -----------------------------------------------------
* MDB2GMS Version 2.1, March 2004
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* DAO version: 3.6
* Jet version: 4.0
* Database:    E:\models\trnsportdata.mdb
* Query:       SELECT city1,city2,distance
*              FROM distances
* -----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----------------------------------------------------
```

The standard export format is to consider the last column the value column and the previous columns as the indices. The indices are separated by a dot, allowing the generated include file to be used as part of a **parameter** declaration statement.

6 Example: Multi-valued tables

Consider the table with two value columns:

DATA

year	loc	Prod	sales	profit
1997	la	Hardware	80	8
1997	la	Software	60	16
1997	nyc	Hardware	110	5
1997	nyc	Software	100	10
1997	sfo	Hardware	80	9
1997	sfo	Software	50	10
1997	was	Hardware	120	7
1997	was	Software	70	20
1998	la	Hardware	70	6
1998	la	Software	70	10
1998	nyc	Hardware	120	7
1998	nyc	Software	120	14

1998	sfo	Hardware	90	12
1998	sfo	Software	70	15
1998	was	Hardware	130	12
1998	was	Software	80	15

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```
SELECT year,loc,prod,sales
FROM data

SELECT year,loc,prod,profit
FROM data
```

If the results are stored in include files **sales.inc** and **profit.inc** then this can be read into GAMS as follows:

```
parameter sales(year,loc,prd) /
$include sales.inc
/;
parameter profit(year,loc,prd) /
$include profit.inc
/;
```

The operation can also be performed in one big swoop by using a different GAMS datastructure:

```
set typ /sales,profit/
parameter data(year,loc,prd,typ) /
$include data.inc
/;
```

This parameter has an extra index **typ** which indicates the data type. To generate the correct include file we can use the following query:

```
SELECT year,loc,prod,'sales',sales
FROM data
UNION
SELECT year,loc,prod,'profit',profit
FROM data
```

The generated include file will look like:

```
* -----------------------------------------------------
* MDB2GMS Version 2.1, March 2004
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* DAO version: 3.6
* Jet version: 4.0
* Database:    E:\models\trnsportdata.mdb
* Query:       SELECT year,loc,prod,'sales',sales
*              FROM data
*              UNION
*              SELECT year,loc,prod,'profit',profit
*              FROM data
* -----------------------------------------------------
1997.la.hardware.profit 8
```

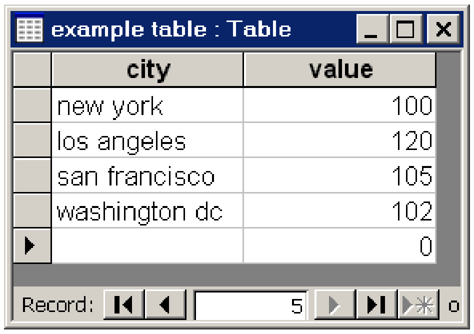
```
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
1998.la.software.profit 10
1998.la.software.sales 70
1998.nyc.hardware.profit 7
1998.nyc.hardware.sales 120
1998.nyc.software.profit 14
1998.nyc.software.sales 120
1998.sfo.hardware.profit 12
1998.sfo.hardware.sales 90
1998.sfo.software.profit 15
1998.sfo.software.sales 70
1998.was.hardware.profit 12
1998.was.hardware.sales 130
1998.was.software.profit 15
1998.was.software.sales 80
* -------------------------------------------------------
```

7 Index mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

```
set i /NY,DC,LA,SF/;
```

Now assume a data table looks like:


example table : Table

city	value
new york	100
los angeles	120
san francisco	105
washington dc	102
	0

This means we have to map ‘new york' to ‘NY' etc. This mapping can be done in two places: either in GAMS or in the database.

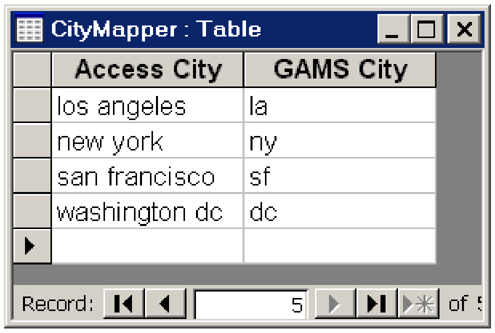
When we export the table directly we get:

```
* -----------------------------------------------------
* MDB2GMS Version 2.1, March 2004
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* DAO version: 3.6
* Jet version: 4.0
* Database:    E:\models\trnsportdata.mdb
* Query:       SELECT city,[value]
*              FROM [example table]
* -----------------------------------------------------
'new york' 100
'los angeles' 120
'san francisco' 105
'washington dc' 102
* -----------------------------------------------------
```

As the index elements contain blanks, the option **Quote Blanks** was used. To import this file and convert it to a different index space we can use the following GAMS code:

```
set i /NY,DC,LA,SF/;

set idb 'from database' /
  'new york',
  'washington dc',
  'los angeles',
  'san francisco'
/;

parameter dbdata(idb) /
$include data.inc
/;

set mapindx(i,idb) /
   NY.'new york'
   DC.'washington dc'
   LA.'los angeles'
   SF.'san francisco'
/;

parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;
```

The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:


CityMapper : Table

Access City	GAMS City
los angeles	la
new york	ny
san francisco	sf
washington dc	dc

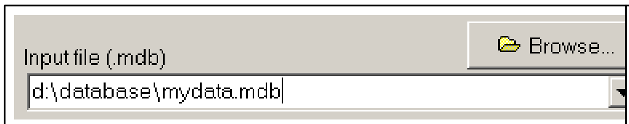
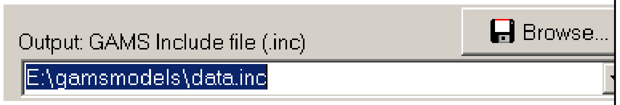
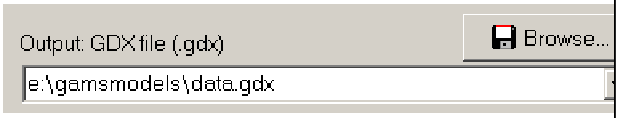
Record: 5 of

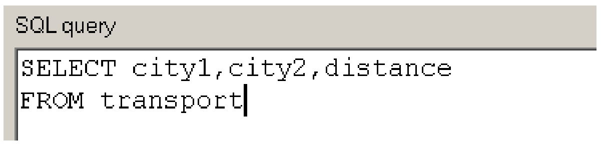
This table can be used in a join to export the data in a format we can use by executing the query:

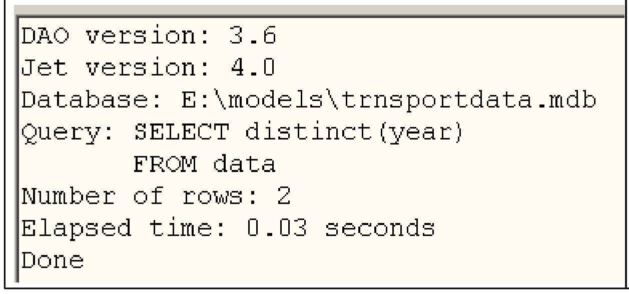
```
SELECT [GAMS City], [value]
FROM [example table],CityMapper
WHERE [Access City]=city
```

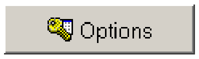
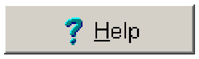
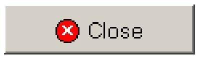
8 Interactive use

When the tool is called without command line parameters, it will startup interactively. Using it this way, one can specify the database file (.MDB file), the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen contains a number of buttons and edit boxes, which are explained below.

 Input file (.mdb) d:\database\mydata.mdb	**Input file (*.MDB).** This is the combo box to specify the input file. The file must be a valid MS Access database file (*.MDB). The browse button can be used to launch a file open dialog which makes it easier to specify a file. The file may be located on a remote machine using the notation *\\machine\directory\file.mdb*.
 Output: GAMS Include file (.inc) E:\gamsmodels\data.inc	**Output GAMS Include file (*.INC)**. If you want to create a GAMS include file, then specify here the destination file. The include file will be an ASCII file that can be read by GAMS using the *$include* command. If the include file already exists, it will be overwritten.
 Output: GDX file (.gdx) e:\gamsmodels\data.gdx	**Output GDX file (*.GDX)** As an alternative to a GAMS include file, the tool can also generate a GDX file. If the GDX file already exists it will be overwritten – it is not possible to append to a GDX file. One or both of the output files need to be specified.

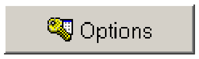
 SQL query
`SELECT city1,city2,distance`
`FROM transport`
	SQL Query The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. The query is passed on directly to the Jet database engine, so the complete power and expressiveness of Access SQL is available. For an exact description of allowed expressions consult a text on MS Access.

 `DAO version: 3.6`
`Jet version: 4.0`
`Database: E:\models\trnsportdata.mdb`
`Query: SELECT distinct(year)`
`       FROM data`
`Number of rows: 2`
`Elapsed time: 0.03 seconds`
`Done`
	Progress Memo. This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.

	The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).
	The **tables button** will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.
	The **options button** will pop up a window where you can specify a number of options.
	The **help button** will show this help.
	If the **OK button** is pressed the query will be executed and an include file or GDX file will be generated.
	Pressing the **batch button** will give information on how the current query can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied to the clipboard. In the IDE press Ctrl-C or choose Edit|Paste to copy the contents of the clipboard to a GAMS text file.
	The **close button** will exit the application. The current settings will be saved in an INI file so when you run MDB2GMS again all current settings will be restored.

9 Options

The **Options** window can be created by pressing the options button:

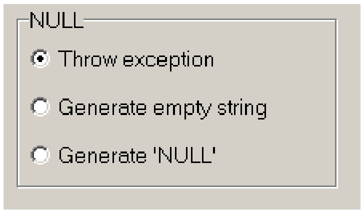
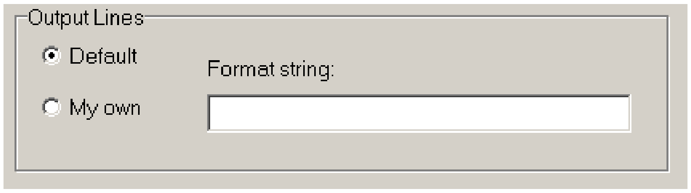


The following options are available in the options window:

	Quote blanks: Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. (In the special case the string contains both, double quotes are replaced by single quotes). For more information see this example. This option only applies to an output include file.
	Mute: Don't include the extra informational text (such as used query etc.) in the include file.

☐ No listing	**No listing:** Surround the include file by *$offlisting* and *$onlisting* so that the data will not be echoed to the listing file. This can be useful for very large data files, where the listing file would become too large to be handled comfortably.
☐ Format SQL	**Format SQL:** If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.

The following options are only needed in special cases:

 NULL: ● Throw exception ○ Generate empty string ○ Generate 'NULL'	NULL: This radio box determines how NULL's are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL's the default to throw an exception is a safe choice. In special cases you may want to map NULL's to an empty string or a 'NULL' string.
 Output Lines: ● Default ○ My own; Format string:	**Output Lines**: By default output lines are created as follows: all first n-1 fields are considered indices and the last n-th column is the value. The format corresponding to this situation is '%s.%s.%s %s' (for a three dimensional parameter). In special cases you may want to tinker with the format string being used. The fields are all considered strings, so only use %s as format placeholder. Make sure you specify exactly the same number of %s's as there are columns in the result set.

The buttons have an obvious functionality:

	The **OK button** will accept the changes made.
	The **Cancel button** wil ignore the changes made, and all option settings will revert to their previous values.
	The **Help button** will show this help text.

10 Batch use

When calling **MDB2GMS** directly from GAMS we want to specify all command and options directly from the command line or from a command file. An example is:

```
C:\tmp> mdb2gms I=C:\tmp\db.mdb O=C:\tmp\data.inc Q=”select i,j,v from t1”
```

This call will perform its task without user intervention. The batch facility can be used from inside a GAMS model, e.g.:

```
parameter c(i,j) 'data from database' /
$call =mdb2gms I=db.mdb O=data.inc Q='select i,j,v from t1'
$include data.inc
/;
```

The $call statement is rather error prone and you will need to spend a little it of time to get the call correct and reliable.

All the possible command line options are listing in command line arguments section. A proper **batch** call will at least contain the following command line parameters:

1. I=inputfilename
2. O=outputincludefile or X=outputgdxfile
3. Q=querystring

If you only specify **I=inputfilename** then the interactive user interface is started with an initial setting of the input file name edit box equal to the name given in the command line argument. Only if an input file, an output file and a query string is provided, the call will be considered a batch invocation.

11 Example

Consider the Access table:

DATA

Year	Loc	prod	sales	Profit
1997	La	hardware	80	8
1997	La	software	60	16
1997	Nyc	hardware	110	5
1997	Nyc	software	100	10
1997	Sfo	hardware	80	9
1997	Sfo	software	50	10
1997	Was	hardware	120	7
1997	Was	software	70	20
1998	La	hardware	70	6
1998	La	software	70	10
1998	Nyc	hardware	120	7
1998	Nyc	software	120	14
1998	Sfo	hardware	90	12
1998	Sfo	software	70	15

1998	Was	hardware	130	12
1998	Was	software	80	15

We want to extract the following information:

- The set **year**
- The set **loc**
- The parameter **sales**
- The parameter **profit**

This can be accomplished using the following GAMS code:

```
set y 'years' /
$call =mdb2gms I=sample.mdb Q="select distinct(year) from data" O=years.inc
$include years.inc
/;

set loc 'locations' /
$call =mdb2gms I=sample.mdb Q="select distinct(loc) from data" O=locations.inc
$include locations.inc
/;

set prd 'products' /hardware, software/;

parameter sales(prd,loc,y) /
$call =mdb2gms I=sample.mdb Q="select prod,loc,year,sales from data" O=sales.inc
$include sales.inc
/;
display sales;

parameter profit(prd,loc,y) /
$call =mdb2gms I=sample.mdb Q="select prod,loc,year,profit from data" O=profit.inc
$include profit.inc
/;
display profit;
```

The $CALL statements assume that **mdb2gms.exe** is in the path. This can be achieved by placing **mdb2gms.exe** in GAMS system directory (the directory where also **gams.exe** is located; the directory can be easily located in the IDE by looking at GAMS executable path in File|Options|Execute). If **mdb2gms.exe** is not in the search path, you can call it explicitly as:

```
$call ="c:\program files\mdb2gms.exe" I=sample.mdb Q="select prod,loc,year,profit from data"
O="d:\data\profit.inc"
```

The resulting listing file will show:

```
   6
   7  set y 'years' /
INCLUDE    E:\models\years.inc
  10  * -----------------------------------------------------
  11  * MDB2GMS Version 2.0, December 2003
  12  * Erwin Kalvelagen, GAMS Development Corp
  13  * -----------------------------------------------------
  14  * DAO version: 3.6
  15  * Jet version: 4.0
  16  * Database:    D:\Win9x Program Files\GAMS21.4\examples\sample.mdb
  17  * Query:       select distinct(year) from data
  18  * -----------------------------------------------------
  19   1997
  20   1998
  21  * -----------------------------------------------------
  22  /;
  23
  24  set loc 'locations' /
INCLUDE    E:\models\locations.inc
  27  * -----------------------------------------------------
  28  * MDB2GMS Version 2.0, December 2003
  29  * Erwin Kalvelagen, GAMS Development Corp
  30  * -----------------------------------------------------
  31  * DAO version: 3.6
  32  * Jet version: 4.0
  33  * Database:    D:\Win9x Program Files\GAMS21.4\examples\sample.mdb
  34  * Query:       select distinct(loc) from data
  35  * -----------------------------------------------------
  36   la
  37   nyc
  38   sfo
  39   was
  40  * -----------------------------------------------------
  41  /;
  42
  43  set prd 'products' /hardware, software/;
  44
  45  parameter sales(prd,loc,y) /
INCLUDE    E:\models\sales.inc
  48  * -----------------------------------------------------
  49  * MDB2GMS Version 2.0, December 2003
  50  * Erwin Kalvelagen, GAMS Development Corp
  51  * -----------------------------------------------------
  52  * DAO version: 3.6
  53  * Jet version: 4.0
  54  * Database:    D:\Win9x Program Files\GAMS21.4\examples\sample.mdb
  55  * Query:       select prod,loc,year,sales from data
  56  * -----------------------------------------------------
  57  hardware.la.1997 80
  58  software.la.1997 60
  59  hardware.nyc.1997 110
  60  software.nyc.1997 100
  61  hardware.sfo.1997 80
  62  software.sfo.1997 50
  63  hardware.was.1997 120
  64  software.was.1997 70
  65  hardware.la.1998 70
  66  software.la.1998 70
  67  hardware.nyc.1998 120
  68  software.nyc.1998 120
  69  hardware.sfo.1998 90
  70  software.sfo.1998 70
  71  hardware.was.1998 130
  72  software.was.1998 80
  73  * -----------------------------------------------------
  74  /;
  75  display sales;
  76
  77  parameter profit(prd,loc,y) /
INCLUDE    E:\models\profit.inc
  80  * -----------------------------------------------------
  81  * MDB2GMS Version 2.0, December 2003
  82  * Erwin Kalvelagen, GAMS Development Corp
  83  * -----------------------------------------------------
  84  * DAO version: 3.6
  85  * Jet version: 4.0
  86  * Database:    D:\Win9x Program Files\GAMS21.4\examples\sample.mdb
  87  * Query:       select prod,loc,year,profit from data
  88  * -----------------------------------------------------
  89  hardware.la.1997 8
  90  software.la.1997 16
  91  hardware.nyc.1997 5
  92  software.nyc.1997 10
  93  hardware.sfo.1997 9
  94  software.sfo.1997 10
  95  hardware.was.1997 7
  96  software.was.1997 20
  97  hardware.la.1998 6
  98  software.la.1998 10
  99  hardware.nyc.1998 7
 100  software.nyc.1998 14
 101  hardware.sfo.1998 12
 102  software.sfo.1998 15
 103  hardware.was.1998 12
 104  software.was.1998 15
 105  * ----------------------------------------------------
 106  /;
 107  display profit;
```

Indeed the includes contain the correct data sets. The include file summary shows that the $CALL and $INCLUDE statements were executed without errors:

```
Include File Summary


   SEQ   GLOBAL TYPE      PARENT   LOCAL  FILENAME

     1        1 INPUT          0       0  D:\Win9x Program Files\GAMS21.4
\examples\salesprofitdb.gms
     2        8 CALL           1       8  =mdb2gms I="D:\Win9x Program Files\GAMS21.4
\examples\sample.mdb" Q="select dis
                                          tinct(year) from data" O=years.inc
     3        9 INCLUDE        1       9  .E:\models\years.inc
     4       25 CALL           1      13  =mdb2gms I="D:\Win9x Program Files\GAMS21.4
\examples\sample.mdb" Q="select dis
                                          tinct(loc) from data" O=locations.inc
     5       26 INCLUDE        1      14  .E:\models\locations.inc
     6       46 CALL           1      20  =mdb2gms I="D:\Win9x Program Files\GAMS21.4
\examples\sample.mdb" Q="select pro
                                          d,loc,year,sales from data" O=sales.inc
     7       47 INCLUDE        1      21  .E:\models\sales.inc
     8       78 CALL           1      26  =mdb2gms I="D:\Win9x Program Files\GAMS21.4
\examples\sample.mdb" Q="select pro
                                          d,loc,year,profit from data" O=profit.inc
     9       79 INCLUDE        1      27  .E:\models\profit.inc
```

Finally the DISPLAY statements show that the data arrived correctly in the required format:

```
----     75 PARAMETER sales

                    1997        1998

hardware.la       80.000      70.000
hardware.nyc     110.000     120.000
hardware.sfo      80.000      90.000
hardware.was     120.000     130.000
software.la       60.000      70.000
software.nyc     100.000     120.000
software.sfo      50.000      70.000
software.was      70.000      80.000



----    107 PARAMETER profit
```

```

                    1997        1998

hardware.la        8.000       6.000
hardware.nyc       5.000       7.000
hardware.sfo       9.000      12.000
hardware.was       7.000      12.000
software.la       16.000      10.000
software.nyc      10.000      14.000
software.sfo      10.000      15.000
software.was      20.000      15.000
```

12 Multi-Query Batch use

In some cases a number of small queries need to be performed on the same database. To do individual calls to **MDB2GMS** can become expensive: there is significant overhead in starting Access and opening the database. For these cases we have added the option to do multiple queries in one call. To write several GAMS include files we can use a command file that looks like:

```
I=sample.mdb

Q1=select distinct(year) from data
O1=year.inc

Q2=select distinct(loc) from data
O2=loc.inc

Q3=select distinct(prod) from data
O3=prod.inc

Q4=select prod,loc,year,sales from data
O4=sales.inc

Q5=select prod,loc,year,profit from data
O5=profit.inc
```

We see that the option **Q*n*** is matched by an option **O*n***. That means that the results of the *n*-th query are written to the *n*-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use:

```
I=sample.mdb
X=sample.gdx

Q1=select distinct(year) from data
S1=year

Q2=select distinct(loc) from data
S2=loc

Q3=select distinct(prod) from data
S3=prd

Q4=select prod,loc,year,sales from data
P4=sales

Q5=select prod,loc,year,profit from data
P5=profit
```

Here we see that a query **Q*n*** is matched by either a set name **S*n*** or a parameter name **P*n***. The name of the GDX file is specified with the **X** option.

For a complete example see section Multi-query Batch example.

13 Multi-Query Batch example

As an example database we use the Access table:

DATA

Year	Loc	Prod	sales	Profit
1997	La	Hardware	80	8
1997	La	Software	60	16
1997	Nyc	Hardware	110	5
1997	Nyc	Software	100	10
1997	Sfo	Hardware	80	9
1997	Sfo	Software	50	10
1997	Was	Hardware	120	7
1997	Was	Software	70	20
1998	La	Hardware	70	6
1998	La	Software	70	10
1998	Nyc	Hardware	120	7
1998	Nyc	Software	120	14
1998	Sfo	Hardware	90	12
1998	Sfo	Software	70	15
1998	Was	Hardware	130	12
1998	Was	Software	80	15

We want to extract the following information:

- The set **year**
- The set **loc**
- The parameter **sales**
- The parameter **profit**

This can be accomplished using the following GAMS code:

```
$ontext

  Example database access with MDB2GMS

  Multiple queries in one call

$offtext

$onecho > cmd.txt
```

```
I=%system.fp%sample.mdb

Q1=select distinct(year) from data
O1=year.inc

Q2=select distinct(loc) from data
O2=loc.inc

Q3=select distinct(prod) from data
O3=prod.inc

Q4=select prod,loc,year,sales from data
O4=sales.inc

Q5=select prod,loc,year,profit from data
O5=profit.inc
$offecho

$call =mdb2gms @cmd.txt

set y years /
$include year.inc
/;
set loc locations /
$include loc.inc
/;
set prd products /
$include prod.inc
/;

parameter sales(prd,loc,y) /
$include sales.inc
/;
display sales;

parameter profit(prd,loc,y) /
$include profit.inc
/;
display profit;
```

The same example imported through a GDX file can look like:

```
$ontext

  Example database access with MDB2GMS

  Multiple queries in one call, store in GDX file

$offtext

$onecho > cmd.txt
I=%system.fp%sample.mdb
X=sample.gdx

Q1=select distinct(year) from data
s1=year

Q2=select distinct(loc) from data
s2=loc

Q3=select distinct(prod) from data
s3=prd

Q4=select prod,loc,year,sales from data
```

```
p4=sales

Q5=select prod,loc,year,profit from data
p5=profit
$offecho

$call =mdb2gms @cmd.txt

$call =gdxviewer sample.gdx

set y 'years';
set loc 'locations';
set prd 'products';
parameter sales(prd,loc,y);
parameter profit(prd,loc,y);

$gdxin 'sample.gdx'
$load y=year loc prd sales profit


display sales;
display profit;
```

The call to **gdxviewer** will display the gdx file in the stand-alone GDX viewer.

14 Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

- The database can change between runs, leading to results that are not reproducible. A possible scenario is a user calling you with a complaint: “the model is giving strange results”. You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.
- There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.
- It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete “under-the-hood” approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

Data extraction --> Data manipulation --> Model definition --> Model solution --> Report writing

we can conveniently use the save/restart facility. The individual submodel are coded as:

Step 0: sr0.gms

```
$ontext

  step 0: data extraction from database
  execute as:  > gams sr0 save=s0

$offtext
```

```
set i 'suppliers';
set j 'demand centers';

parameter demand(j);
parameter supply(i);
parameter dist(i,j) 'distances';

$onecho > cmd.txt
I=%system.fp%transportation.mdb

Q1=select name from suppliers
O1=i.inc

Q2=select name from demandcenters
O2=j.inc

Q3=select name,demand from demandcenters
O3=demand.inc

Q4=select name,supply from suppliers
O4=supply.inc

Q5=select supplier,demandcenter,distance from distances
O5=dist.inc
$offecho

$call =mdb2gms.exe @cmd.txt

set i /
$include i.inc
/;

set j /
$include j.inc
/;

parameter demand /
$include demand.inc
/;

parameter supply /
$include supply.inc
/;

parameter dist /
$include dist.inc
/;

display i,j,demand,supply,dist;
```

Step 1: sr1.gms

```
$ontext

  step 1: data manipulation step
  execute as:  > gams sr1 restart=s0 save=s1

$offtext

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 * dist(i,j) / 1000 ;
```

Step 2: sr2.gms

```
$ontext

  step 2: model definition
  execute as:  > gams sr2 restart=s1 save=s2

$offtext

Variables
     x(i,j)  'shipment quantities in cases'
     z       'total transportation costs in thousands of dollars' ;

Positive Variable x ;

Equations
     ecost        'define objective function'
     esupply(i)   'observe supply limit at plant i'
     edemand(j)   'satisfy demand at market j' ;

ecost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

esupply(i) ..   sum(j, x(i,j))  =l=  supply(i) ;

edemand(j) ..   sum(i, x(i,j))  =g=  demand(j) ;
```

Step 3: sr3.gms

```
$ontext

  step 3: model solution
  execute as:  > gams sr3 restart=s2 save=s3

$offtext

option lp=cplex;

Model transport /all/ ;
Solve transport using lp minimizing z ;
```

Step 4: sr4.gms

```
$ontext

  step 4: report writing
  execute as:  > gams sr4 restart=s3

$offtext

abort$(transport.modelstat <> 1) "model not solved to optimality";

display x.l,z.l;
```

A model that executes all steps can be written as:

```
execute '=gams.exe sr0 lo=3 save=s0';
abort$errorlevel "step 0 failed";
```

```
execute '=gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorlevel "step 1 failed";

execute '=gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorlevel "step 2 failed";

execute '=gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorlevel "step 3 failed";

execute '=gams.exe sr4 lo=3 restart=s3';
abort$errorlevel "step 4 failed";
```

If you only change the reporting step, i.e. generating some output using PUT statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

MDB2GDX1.GMS:
```
$ontext

   Store data from Access database into a GDX file.

$offtext

execute '=gams.exe sr0 lo=3 gdx=trnsport.gdx';
abort$errorlevel "step 0 failed";

execute '=gdxviewer.exe trnsport.gdx';
```

We can also let MDB2GMS create the GDX file:

MDB2GDX2.GMS:
```
$ontext

   Store data from Access database into a GDX file.

$offtext

$onecho > cmd.txt
I=%system.fp%transportation.mdb
X=%system.fp%transportation.gdx

Q1=select name from suppliers
S1=i

Q2=select name from demandcenters
S2=j

Q3=select name,demand from demandcenters
P3=demand

Q4=select name,supply from suppliers
```

```
P4=supply

Q5=select supplier,demandcenter,distance from distances
P5=dist
$offecho

$call =mdb2gms.exe @cmd.txt
```

The first approach has the advantage that a complete audit record is available from the data moved from the database to the GDX file in the **sr0.lst** listing file. If someone ever wonders what came out of the database and how this was stored in the GDX file, that file gives the answer.

To load the GDX data the following fragment can be used:

GDXTRNSPORT.GMS:

```
$ontext

   Load transportation data from GDX file

   Compile time loading

$offtext

set i 'suppliers';
set j 'demand centers';

parameter demand(j);
parameter supply(i);
parameter dist(i,j) 'distances';

$gdxin transportation.gdx
$load i j demand supply dist


display i,j,demand,supply,dist
```

In one application I had to retrieve data from the database each morning, at the first run of the model. The rest of the day, the data extracted that morning could be used. The following logic can implement this:

```
$ontext

    Retrieve data from data base first run each morning.

$offtext

$onecho > getdate.txt
I=%system.fp%transportation.mdb
Q=select day(now())
O=dbtimestamp.inc
$offecho

$if not exist dbtimestamp.inc $call "echo 0 > dbtimestamp.inc"

scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc
/;

scalar currentday 'day of this run';
currentday = gday(jnow);
```

```
display "compare", dbtimestamp,currentday;

if (dbtimestamp<>currentday,

   execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
   abort$errorlevel "step 0 (database access) failed";

   execute '=mdb2gms.exe @getdate.txt'
);
```

The include file **dbtimestamp.inc** contains the day of the month (1,..,31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the **dbtimestamp.inc** file. This last operation could be done using a PUT statement, but in this case we used an SQL statement.

15 Command-line Arguments

I=inputfile	This option is required and specifies the name of the .MDB file containing the Access database. If the file contains blanks the name should be surrounded by double quotes. It is advised to use absolute paths, so Access has no confusion what file to open. On a network UNC names can be used, and files from another computer can be accessed, e.g. "\\hostname\c\my documents\a.mdb." This option is required for batch processing. To specify a path equal to the location where the .gms file is located, you can use: **I=**%system.fp%mydb.mdb
O=outputincludefile	This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter statement. Make sure the directory is writable. UNC names can be used. An output file must be specified for batch operation: i.e. either O= or X= needs to be specified (or both).
O*n*=outputincludefile	When multiple queries are used, you can append a number to match a query with an output file: **Q2=**"select a,b from table" **O2=**ab.inc See section Multi-query Batch use
X=outputGDXfile	This option specifies the name of the output file. The format of the output file will be a GAMS GDX file. Make sure the directory is writable. UNC namescan be used. An output file must be specified for batch operation: i.e. either O= or X= needs to be specified (or both).

Q=query	This option can be used to specify an SQL query. Queries contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by Access we refer to the documentation that comes with MS Access. One notable syntax feature is that when field names or table names contain blanks, they can be specified in square brackets. Examples:
	
	`Q="select * from mytable"`
	`Q="select year, production from [production table]"`

	Q=`"select [GAMS City],value from [example table], CityMapper where [Access City]=city"`
Q*n*=query	When multiple queries are used, you can append a number to match a query with an output file: **Q2=**`"select a,b from table"` **O2=**`ab.inc` See section Multi-query Batch use
S=setname	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.
S*n*=setname	If multiple queries are used, you can append a number to match the query: **Q2**=`"select i from table"` **S2**=`I` See section Multi-query Batch use
Y=setname	Specify the name of the set to be written and assume that the set contains element texts (string values).
Y*n*=setname	If multiple queries are used, you can append a number to match the query: **Q2**=`"select i from table"` **Y2**=`I` See section Multi-query Batch use
P=parametername	If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file.
P*n*=parametername	If multiple queries are used, you can append a number to match the query: **Q2**=`"select i,v from table"` **P2**=`v` See section Multi-query Batch use
D	Debug. This option can be used for debugging purposes. If specified the import filter will no run minimized but "restored", i.e. as a normal window. In addition the program will not terminate until the user clicks the Close button. This allows you to monitor possible errors during execution of **mdb2gms**.
B	If this parameter is specified, strings that have blanks in them will be quoted. If the string is already quoted this step is not performed. If the name contains an embedded single quote, the surrounding quotes will be double quotes. If the name already contains a double quote, the surrounding quotes will be single quotes. If both single and double quotes are present in the string, then all double quotes are replaced by single quotes and the surrounding quotes will be double quotes. By default this option is turned off.
M	Run in modest or mute mode: no additional information, such as version number etc. is written to the listing file.
L	Embed the data in $*offlisting*, $*onlisting*. A quick way to reduce the size of the listing file.
@filename **@"file name"**	Causes the program to read options from a file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if it were specified on the command line.
N=inifilename	Use a different Inifile than the standard **mdb2gms.ini** located in the

	same directory as the executable **mdb2gms.exe**.
F*n*=formatstring	In special cases we can apply a format string on the include file output (not for GDX output). Each column in the result set is a string and can be represented by a %s in the format string.

16 $CALL command

The **$CALL** command in GAMS will execute an external program at compile time. There are two forms:

```
       $call externalprogram
       $call =externalprogram
```

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=', bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form `$call externalprogram` which uses the command processor.
2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form `$call externalprogram`
3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form `$call =externalprogram`. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: `$call =externalprogram`.
4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: `$call start /w externalwindowsprogram`. Otherwise, it is preferred to use: `$call =externalwindowsprogram`.

In general it is recommended to use the `$call =externalprogram` version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
       $call externalprogram parameter1 parameter2
       $call =externalprogram parameter1 parameter2
```

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
       $call '"external program" "parameter 1" "parameter 2"'
       $call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes *only if the = appears outside the double quotes*.

17 Command files

Parameters can be specified in a command file. This is important if the length of the command line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command lines. Instead of specifying a long command line as in:

```
$call =mdb2gms I="c:\My Documents\test.mdb" O="c:\My Documents\data.inc" Q="Select * from
mytable"
```

we can use a command line like:

```
$call =mdb2gms @"c:\My Documents\options.txt"
```

The command file **c:\My Documents\options.txt** can look like:

```
I=c:\My Documents\test.mdb
O=c:\My Documents\data.inc
Q=Select * from mytable
```

It is possible to write the command file from inside a GAMS model using the $echo command. The following example will illustrate this:

```
$set cmdfile "c:\windows\temp\commands.txt"
$echo "I=E:\models\labordata.mdb" >  "%cmdfile%"
$echo "O=E:\models\labor.INC"     >> "%cmdfile%"
$echo "Q=select * from labor"     >> "%cmdfile%"
$call =mdb2gms @"%cmdfile%"
parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

Newer versions of GAMS allow:

```
$set cmdfile "c:\windows\temp\commands.txt"
$onecho  >  "%cmdfile%"
I=E:\models\labordata.mdb
O=E:\models\labor.INC
Q=select * from labor
$offecho
$call =mdb2gms @"%cmdfile%"
parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character \ as the last character. E.g.:

```
Q=sel ect  pr od, l oc, year , ' sal es' , sal es  f r om dat a \
  uni on \
  sel ect  pr od, l oc, year , ' pr of i t ' , sal es  f r om dat a
```

18 Notes

GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: $GDXIN, $LOAD, EXECUTE_LOAD, EXECUTE_UNLOAD. The GDX=filename command line option will save all data to a GDX file. A GDX file can be viewed in the IDE using *File|Open*.

UNC Names

UNC means Unified Naming Convention. UNC names are a microsoft convention to name files across a network. The general format is:

\\<server>\<share>\<path>\<file>

Examples:

\\athlon\c\My Documents\mdb2gms.rtf

Quotes

Examples of handling of indices when the option "quote blanks" is used:

Input	**output**	**remarks**
Hello	hello	No blanks or embedded quotes
"hello"	"hello"	not touched, is quoted already
'hello'	'hello'	id.
"hello'	"hello'	id, but will generate an error in GAMS
o'brien	"o'brien"	
'o'brien'	'o'brien'	not touched, will generate an error in GAMS
art"ificial	'art"ificial'	
art"ifi'cial	"art'ifi'cial"	

Compile time commands

All **$** commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed **before** an execution time command, even if it is below. As an example consider:

```
      file batchfile /x.bat/;
      put patchfile;
      putclose "dir"/;
      $call x.bat
```

This fragment does not work correctly as already during compilation, the $call is executed, while the put statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

```
       $echo "dir" > x.bat
       $call x.bat
```

or by moving the external program invocation to execution time:

```
       file batchfile /x.bat/;
       put patchfile;
       putclose "dir"/;
       execute x.bat;
```

Notice that all **$** commands do not include a semi-colon but are terminated by the end-of-line.