

# 1 Contents

## GDXVIEWER

- Overview
- Requirements
- Creating GDX files
- Viewing GDX files
- Exporting an identifier
  - Exporting to a text file
  - Exporting to a CSV file
  - Exporting to an XLS file
  - Exporting to an XLS pivot table
  - Exporting to a GAMS include file
  - Exporting to an Access table
  - Exporting to an SQL table
  - Exporting to MS SQL Server
  - Exporting an SQL Insert script
  - Exporting an SQL Update script
  - Exporting HTML
  - Exporting XML
  - Export fields
  - Special values
- Plotting data
- Cube view
  - Exporting cubes
- Command line operation

# 2 Overview

**GDXVIEWER** is a tool to view and convert data contained in GDX files.

Besides inspecting a GDX file, `gdxviewer` allows you to export to a large number of data formats, including ASCII text, CSV, HTML, XML, database, and spreadsheet formats.

This tool is designed as an interactive Windows program, but it can also be operated through command line parameters.

# 3 Requirements

**GDXVIEWER** runs only on PC's running Windows (95/98/NT/XP). The DLL **GDXIO.DLL** needs to be in the same location as **GDXVIEWER.EXE**. If XLS files are saved, MS Excel needs to be present. If MDB database files are saved, MS Access needs to be present.

If **GDXIO.DLL** is not found in the same directory as the executable `gdxviewer.exe`, the following window will be shown:



A simple way to make sure that GDXVIEWER has access to the GDXIO.DLL dynamic load library is to place **gdxviewer.exe** (and **gdxviewer.chm**) in the GAMS system directory, e.g. **c:\program files\GAMS21.3**.

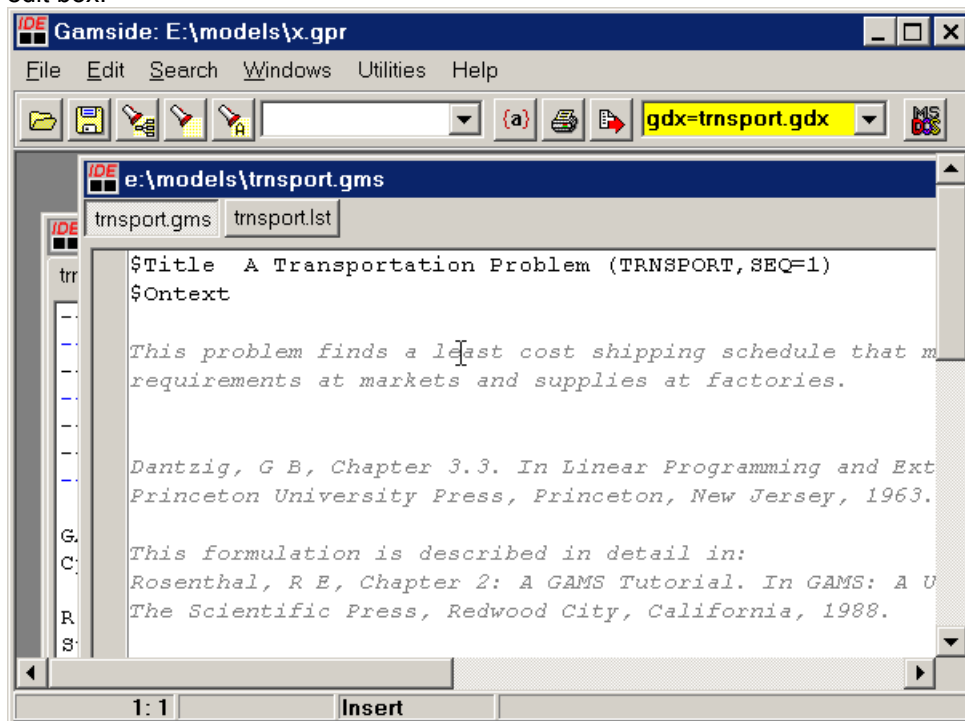
## 4 Creating GDX files

GDX files are binary data files. They can contain sets, parameters (including scalars), equations and variables. These files can be generated by a number of tools: by GAMS itself, by utilities such as MDB2GMS, SQL2GMS, GDXXRW.

To save all data from a GAMS model into a GDX file you can use the **GDX=fln** command line parameter:

```
C:\> gams trnsport.gms GDX=trnsport.gdx
```

From the IDE you can specify the command line parameter **GDX=trnsport.gdx** in the parameter edit box:



To selectively place identifiers in a GDX file you can use the **execute\_unload** statement:

```
Model transport /all/ ;
Solve transport using lp minimizing z ;
Display x.l, x.m ;

execute_unload 'results.gdx', i, j, x;
```

In this example the sets **i** and **j** and the variable **x** are saved to the GDX file **results.gdx**.

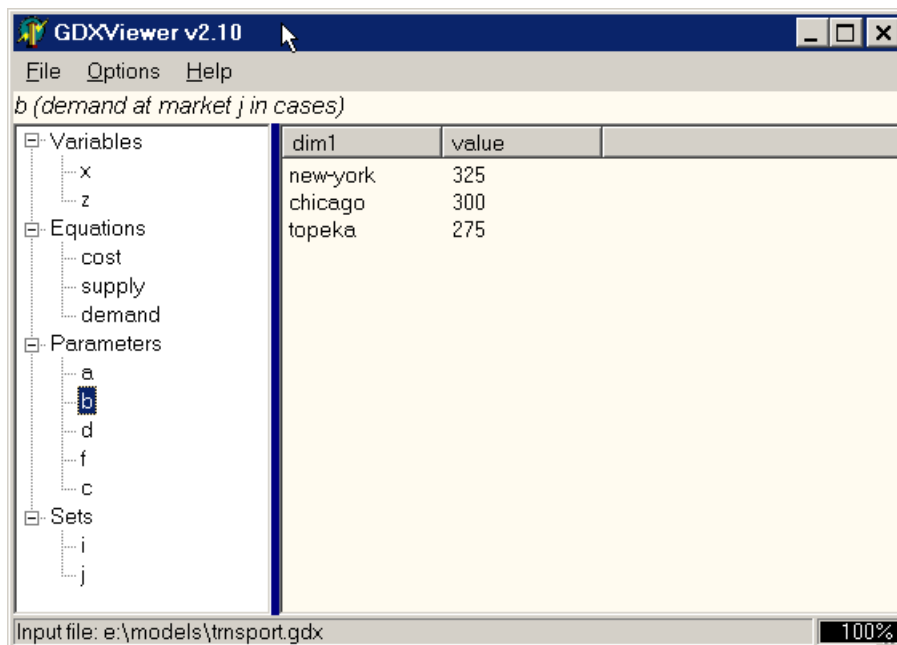
Other ways to create GDX files include:

- The **MDB2GMS** tool can be used to convert data stored in MS Access tables to GDX files
- The **SQL2GMS** tool can read data from virtually any SQL database (including any ODBC accessible database) and can create GDX files.
- The tool **GDXXRW** allows data from an Excel spreadsheet to be stored in a GDX file (see: <http://www.gams.com/contrib/GDXUtils.pdf>).
- **\$GDXOUT** allows you to write data to a GDX file during GAMS compile time. This is not as useful as `execute_unload` but may have its use in special cases.
- You can write your own program to write a GDX file. There is an API and bindings for different languages such as Delphi, Kylix, VB6, VBA, VB.NET, C/C++, C#, Java, Fortran (see <http://www.gams.com/~erwin/interface/interface.html>).

See also: <http://www.gams.com/mccarl/gdxuseage.htm>.

## 5 Viewing GDX files

After loading a GDX file in **gdxviewer** the content of the file is displayed in list view. The left-hand side of the window shows the index of the GDX file organized in a tree structure. When clicking on an identifier, the right-hand-side will display the actual data for the identifier.



When variables are shown, more information is available, such as bounds (lower and upper bounds) and marginals.

The GDX file can be loaded interactively using the *File|Open* menu, or it can be launched from the command line:

```
C:\> gdxviewer e:\models\trnsport.gdx
```

The command line specification can also be used to launch gdxviewer from within a GAMS model as in:

```

Model transport /all/ ;
Solve transport using lp minimizing z ;
Display x.l, x.m ;

execute_unload 'results.gdx', i, j, x;
execute '=gdxviewer results.gdx';

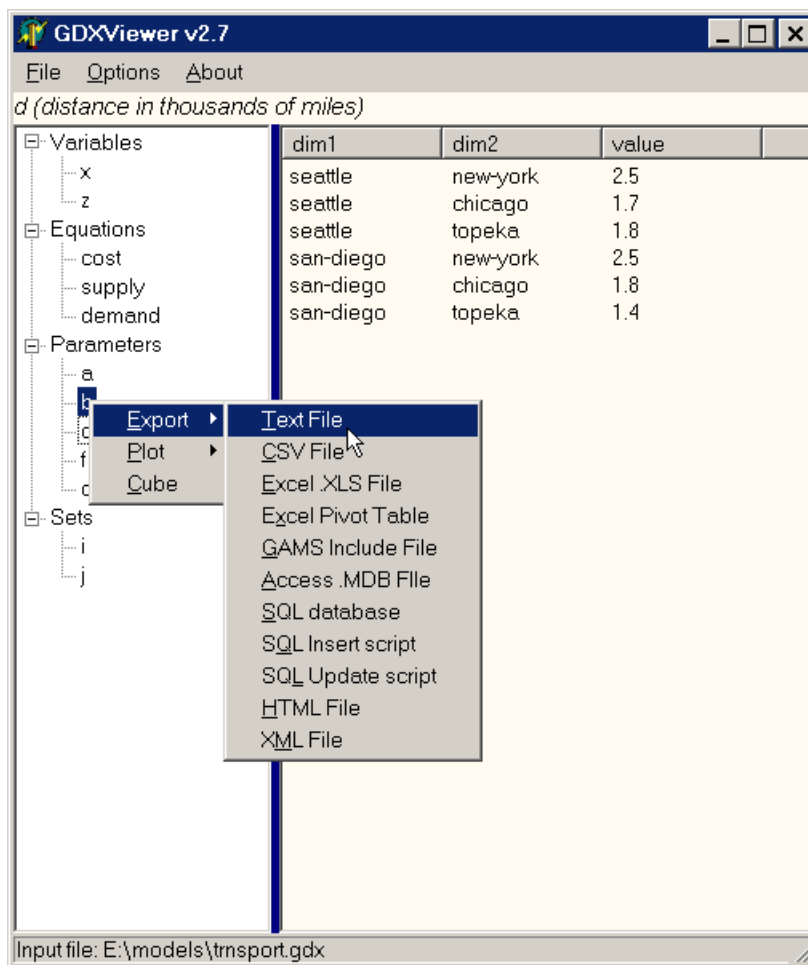
```

In this case **gdxviewer.exe** was located in the GAMS system directory, such that **execute** had no problems in finding it.

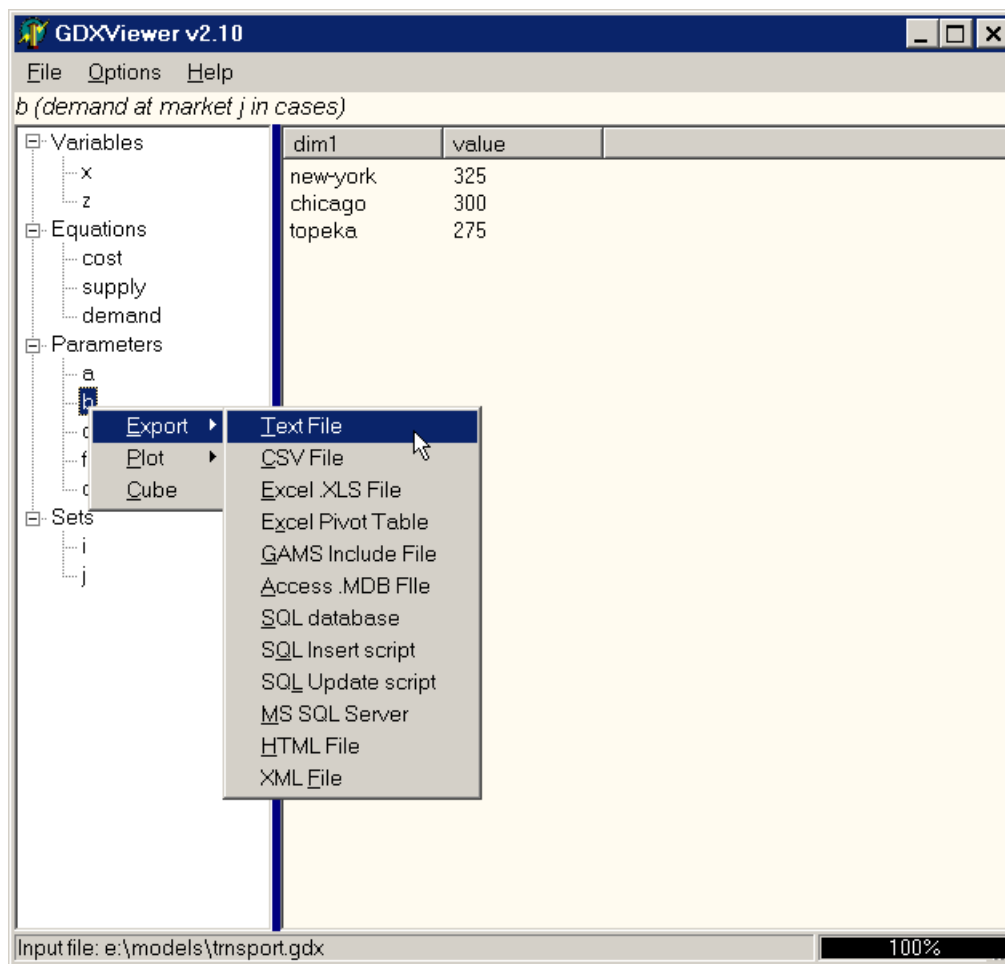
Note: there are alternative tools to view GDX files. The GAMS IDE has a built-in GDX file viewer (use *File|Open*) and there is a command line utility called **GDXDUMP** (see <http://www.gams.com/contrib/GDXUtils.pdf>)

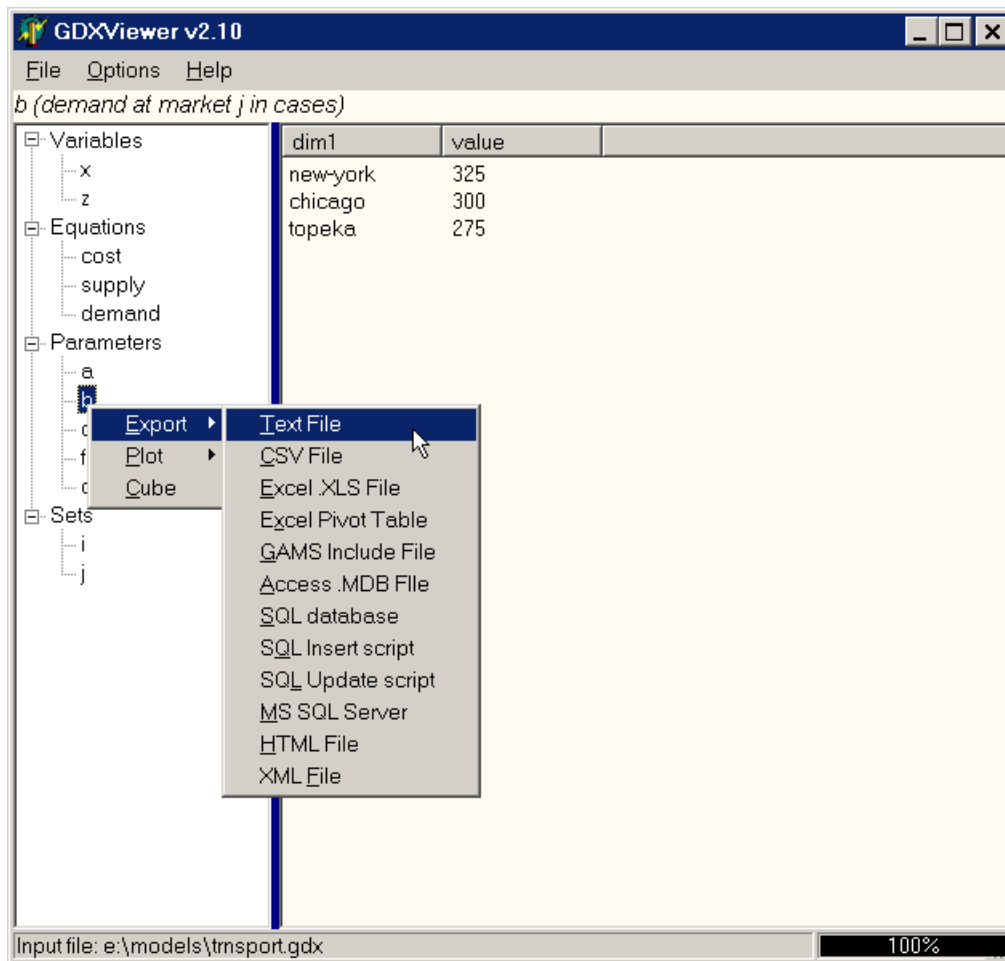
## 6 Exporting an identifier

When the right mouse button is clicked on an identifier a pop-up menu is presented that allows you to export an identifier to a number of target formats.



The same operation can be invoked from the *File|Export* menu:



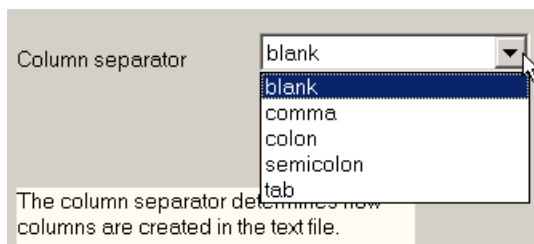


## 7 Exporting to a text file

The **text file** export facility (*File|Export|Text File*) will write a GAMS identifier to a standard ASCII text file. Such a text file can look like:

```
seattle new-york 2.5
seattle chicago 1.7
seattle topeka 1.8
san-diego new-york 2.5
san-diego chicago 1.8
san-diego topeka 1.4
```

The separator symbol can be set using the menu *Options|Configuration|Text File*:



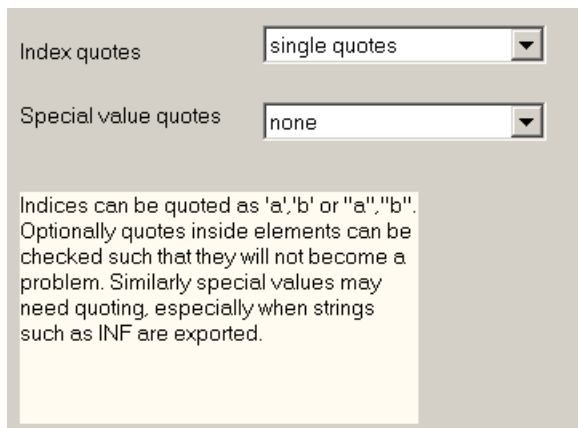
Other options that involve the format of the text file being written are: *Options|Configuration|Export* and *Options|Configuration|Special Values*. Currently there are no facilities to write fixed format text files. If you need to write fixed format text files you can use the GAMS PUT statement.

## 8 Exporting CSV files

Comma-separated Values (*File|Export|CSV File*) is a popular format to exchange data between applications. An example of such a file is:

```
' new-york' , 325  
' chicago' , 300  
' topeka' , 275
```

Strings are surrounded by quotes and each field is separated by a comma. The precise format can be specified using the menu *Options|Configuration|CSV File*:



Index quotes: single quotes

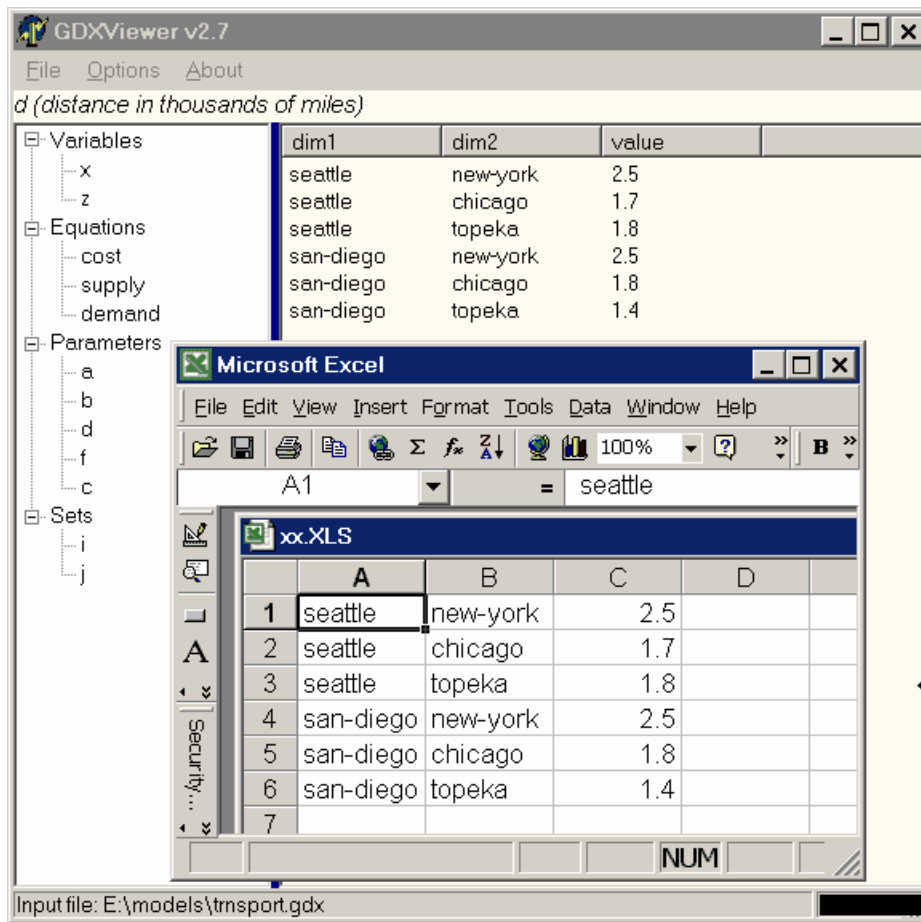
Special value quotes: none

Indices can be quoted as 'a','b' or "a","b".  
Optionally quotes inside elements can be checked such that they will not become a problem. Similarly special values may need quoting, especially when strings such as INF are exported.

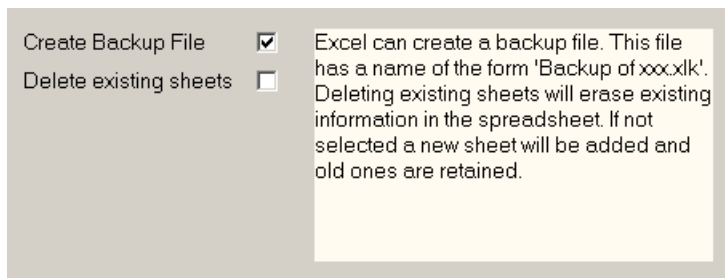
Other options that involve the format of the CSV file being written are: *Options|Configuration|Export* and *Options|Configuration|Special Values*.

## 9 Exporting XLS files

A GAMS identifier can be exported directly to an MS Excel spreadsheet using *File|Export|Excel XLS File*:



There are a few options available for this operation. Under *Options|Configuration|Excel* the following settings can be changed:



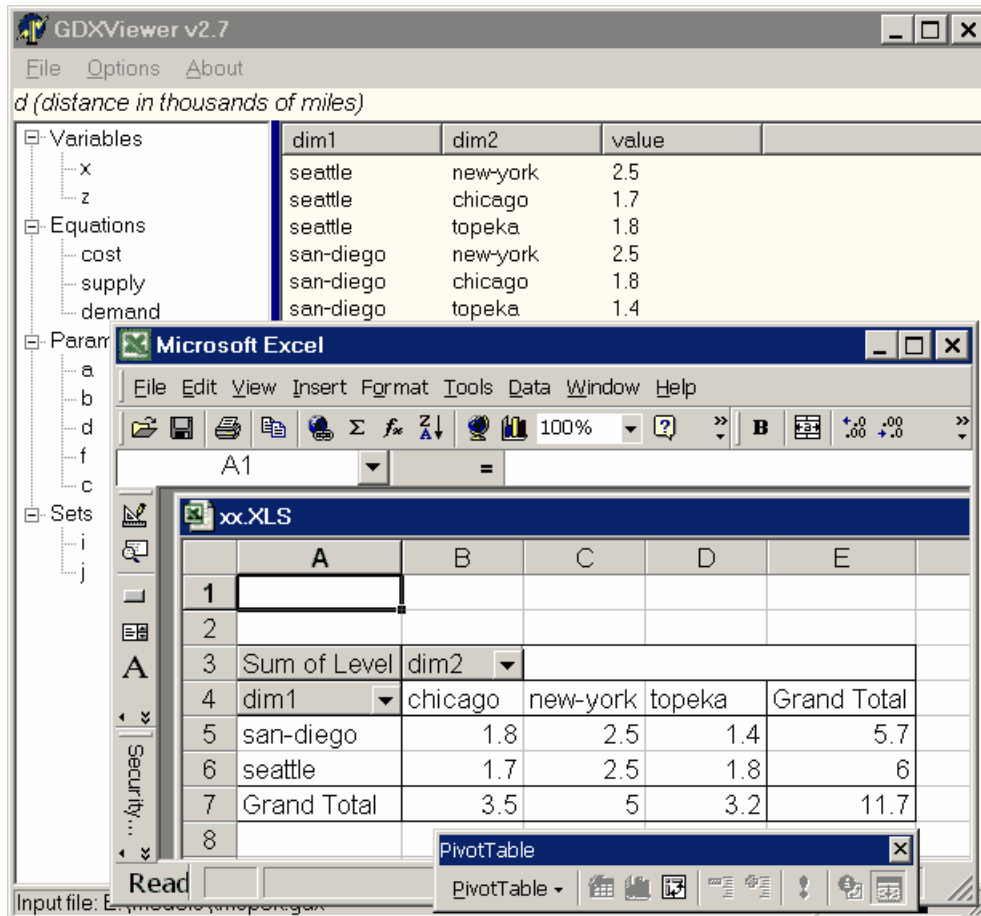
Other options that involve the format of the XLS file being written are: *Options|Configuration|Export* and *Options|Configuration|Special Values*. Exporting to Excel is only available if you have Microsoft Excel installed on your machine.

Note: We can write all symbols in the.gdx file to Excel by specifying ID=\* on the command line.



## 10 Exporting pivot tables

We can export to an XLS file and create a Pivot Table automatically (*File|Export|Excel Pivot Table*):



Pivot tables are a very convenient way to analyze multi-dimensional data.

The following options are available: *Options|Configuration|Excel*, *Options|Configuration|Export* and *Options|Configuration|Special Values*.

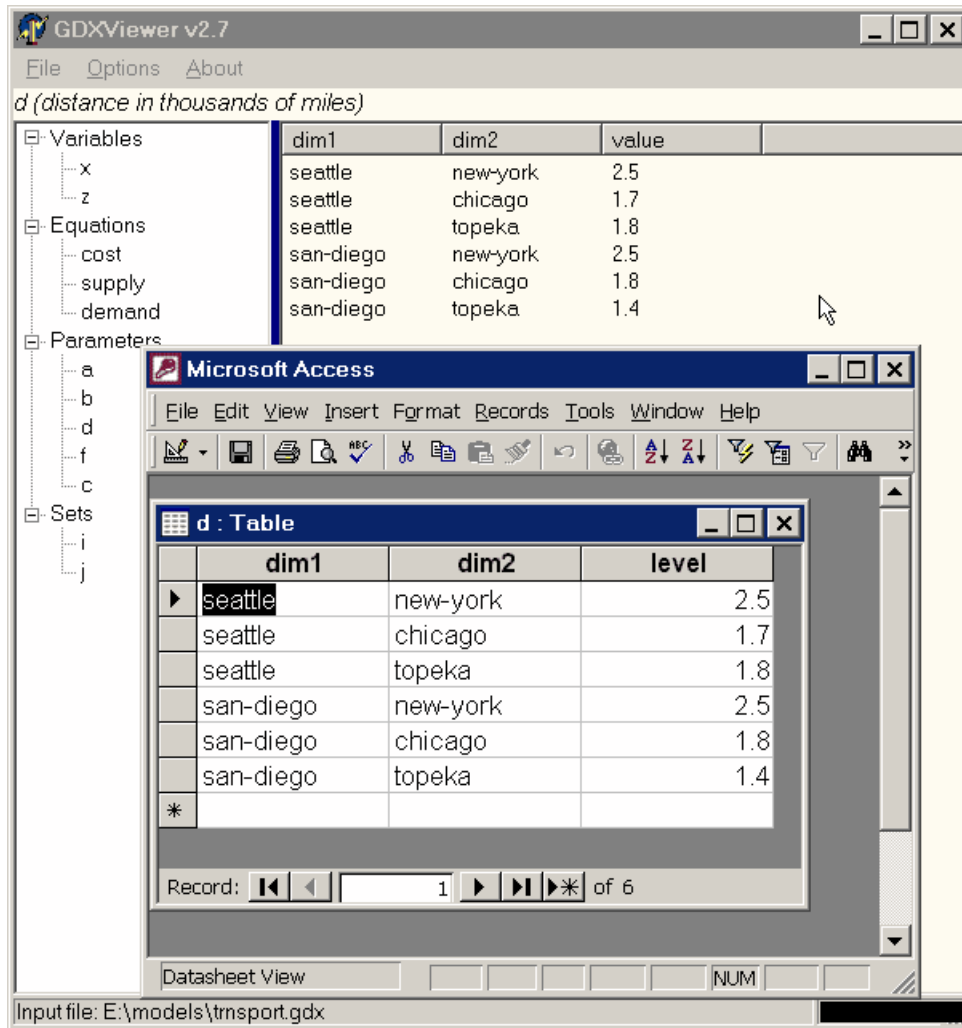
## 11 Exporting GAMS include files

The option *File|Export|GAMS Include file* will export an identifier to a GAMS include file format. An example of such an exported include file can look like:

```
PARAMETER d "distance in thousands of miles" /
seattle.new-york 2.5
seattle.chicago 1.7
seattle.topeka 1.8
san-diego.new-york 2.5
san-diego.chicago 1.8
san-diego.topeka 1.4
/;
```

## 12 Exporting Access Table

GDXVIEWER can export data directly to a table in an Access database (MDB file) using *File|Export|Access MDB File*. The name of the table will be the name of the parameter. If the table already exists, GDXVIEWER will try to create a new table with a slightly different name (e.g. d2, d3,...).



An option *Options|Configuration|Access* allows you to set the length of the text fields where the GAMS indices are stored. This length is used when creating the table.

Max length of indices

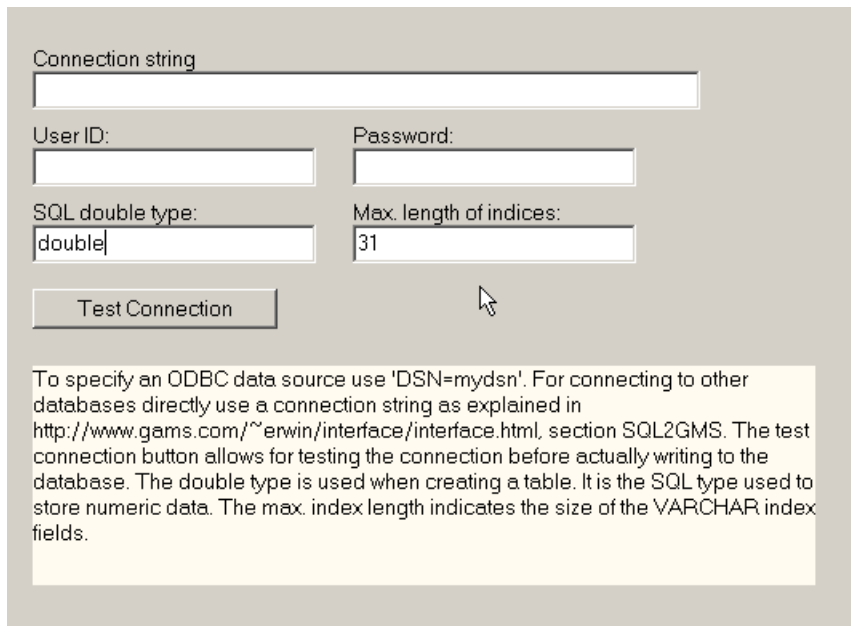
Access import method Indirect (CSV files)  
Direct (INSERT statements)  
Indirect (CSV files)

The indices are stored in text fields. The length of the text fields needs to be large enough so that all indices can be stored. The program will throw an exception if an index is found that does not fit in the text field. The export can be done directly using INSERT statements or via intermediate CSV files. It may come as a surprise that the direct approach is slower.

A feature added in version 2.9 is the possibility to use intermediate CSV (comma separated value) files instead of using direct SQL INSERT statements. The CSV files can be read into Access using a bulk operation and is therefore faster for large datasets. When using CSV files make sure double quotes are used (if single quotes are used they will become part of the data). The temporary CSV files will be written to the Windows TEMP directory (e.g. C:\WINDOWS\TEMP). When the import is done, these scratch files will be removed automatically. If you want to look at the CSV files that are being fed into Access, export the data to a CSV file.

## 13 Exporting SQL Table

It is possible to export data to SQL databases through ADO which includes all databases accessible through ODBC. The configuration information can be specified in *Options|Configuration|SQL Database*.



Connection string

User ID: Password:

SQL double type: double Max. length of indices: 31

Test Connection

To specify an ODBC data source use 'DSN=mydsn'. For connecting to other databases directly use a connection string as explained in <http://www.gams.com/~erwin/interface/interface.html>, section SQL2GMS. The test connection button allows for testing the connection before actually writing to the database. The double type is used when creating a table. It is the SQL type used to store numeric data. The max. index length indicates the size of the VARCHAR index fields.

The **Test Connection** button will allow you to check the configuration and see if the database can be connected to.

The SQL data for double precision number is not always the same for each database. E.g. for MS Access you can use **double** while for MS SQL server you can use **float**.

When exporting data a new table is created with the name of the identifier. If such a table already exists, names like **name2**, **name3**, are tried.

## 14 Exporting to MS SQL Server

We can export to Microsoft SQL Server through the standard SQL export facility. However a special facility called BULK INSERT is only available through the specialized SQL Server export tool. BULK INSERT writes a TAB delimited text file to the Windows TEMP directory and subsequently calls BULK INSERT to load that file. This way is often much faster than using individual INSERT statements for each record.

## 15 Exporting SQL Insert script

An SQL script with INSERT statements like:

```
INSERT INTO dist(city1,city2,distance) VALUES('seattle','new-york',2.5);
INSERT INTO dist(city1,city2,distance) VALUES('seattle','chicago',1.7);
INSERT INTO dist(city1,city2,distance) VALUES('seattle','topeka',1.8);
INSERT INTO dist(city1,city2,distance) VALUES('san-diego','new-york',2.5);
INSERT INTO dist(city1,city2,distance) VALUES('san-diego','chicago',1.8);
INSERT INTO dist(city1,city2,distance) VALUES('san-diego','topeka',1.4);
```

can be generated with *File|Export|SQL Insert script*. The following settings in *Options|Configuration|SQL Insert* were used:

DB Columns	GDX column:	Quote:
city1	dim1	<input checked="" type="checkbox"/>
city2	dim2	<input checked="" type="checkbox"/>
distance	level	<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>

Table name:

☒ Semicolon separator

SQL INSERT statements will be generated of the form: INSERT INTO tablename(dbcol1,dbcol2,...) VALUES (gdxcol1,gdxcol2,...). Column values can be quotes and INSERT statements are often terminated with a semicolon.

## 16 Exporting SQL Update script

An SQL script with UPDATE statements like:

```
UPDATE dist SET distance=2.5 WHERE city1='seattle' AND city2='new-york';
UPDATE dist SET distance=1.7 WHERE city1='seattle' AND city2='chicago';
UPDATE dist SET distance=1.8 WHERE city1='seattle' AND city2='topeka';
UPDATE dist SET distance=2.5 WHERE city1='san-diego' AND city2='new-york';
UPDATE dist SET distance=1.8 WHERE city1='san-diego' AND city2='chicago';
UPDATE dist SET distance=1.4 WHERE city1='san-diego' AND city2='topeka';
```

can be generated with *File|Export|SQL Update script*. The following settings in *Options|Configuration|SQL Update* were used:

DB Columns	GDX column:	Quote:
city1	dim1	<input checked="" type="checkbox"/>
city2	dim2	<input checked="" type="checkbox"/>
distance	level	<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>
		<input type="checkbox"/>

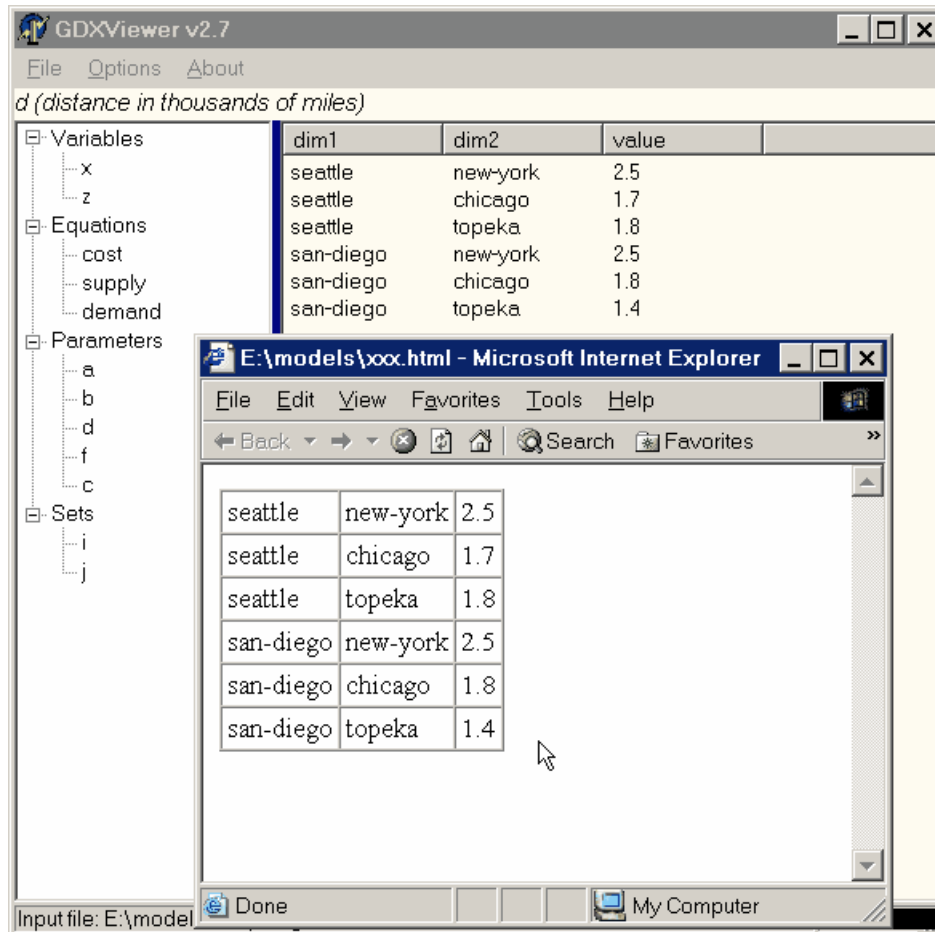
Table name:

☒ Semicolon separator

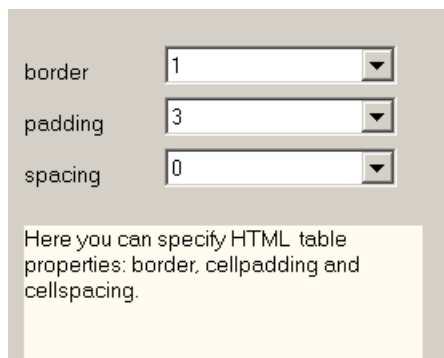
SQL UPDATE statements will be generated of the form: UPDATE tablename SET dbcol1=gdxval1,dbcol2=gdxval2 WHERE dbcol3=gdxdim1 and dbcol4=gdxdim2. Column values can be quotes and UPDATE statements are often terminated with a semicolon.

## 17 Exporting HTML

GDXVIEWER can write an identifier to an HTML file using *File|Export|HTML File*.

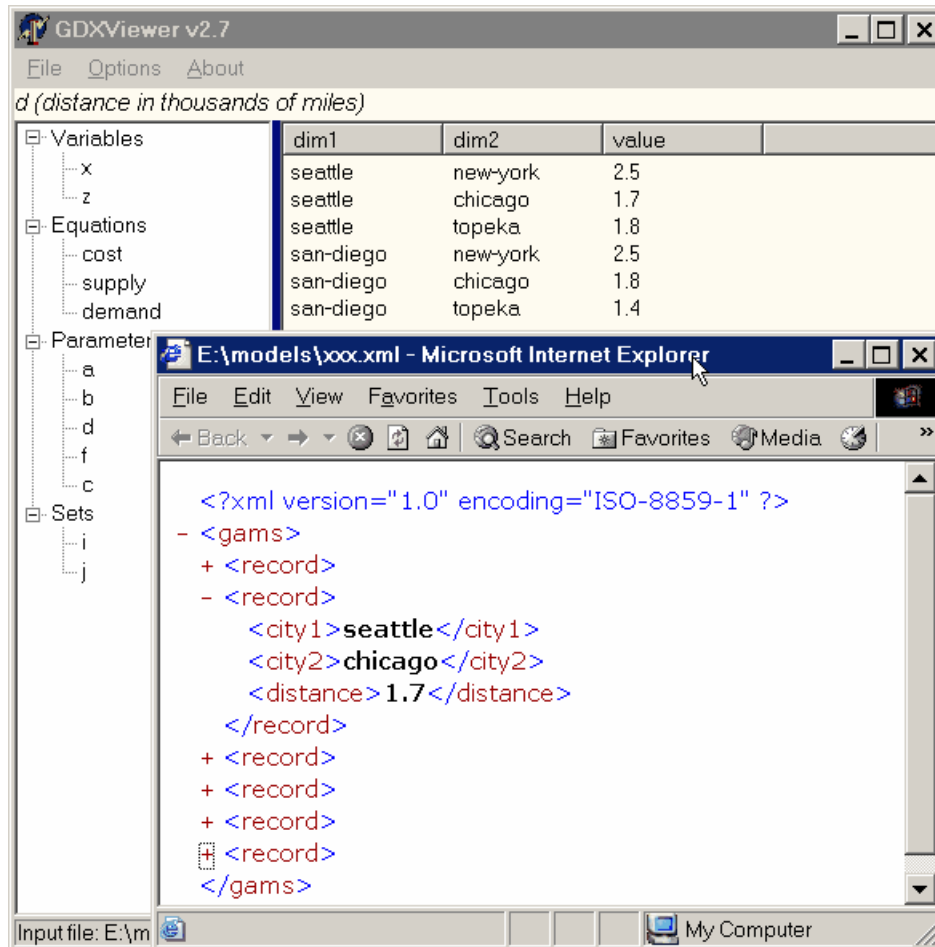


The options relevant to this format are specified in *Options|Configuration|HTML*.



## 18 Exporting XML

GDXVIEWER can write an identifier to an XML file using *File|Export|XML File*.



The XML tags can be specified in *Options|Configuration|XML*.

The screenshot shows the 'index tags' section of the GDXVIEWER configuration window. It contains two main groups of input fields:

- Left Group:**
  - root:** A text box containing 'gams'.
  - record:** A text box containing 'record'.
  - Parameter:** A section containing a **value** text box with 'distance'.
  - Equation/Variable:** A section containing four text boxes: **lo** (lo), **level** (level), **up** (up), and **marginal** (marginal).
- Right Group (index tags):** A list of ten dimension tags, each with a corresponding text box:
  - dim1: city1
  - dim2: city2
  - dim3: dim3
  - dim4: dim4
  - dim5: dim5
  - dim6: dim6
  - dim7: dim7
  - dim8: dim8
  - dim9: dim9
  - dim10: dim10

Below the input fields, a yellow box contains the text: "Specification of the XML tag names."

## 19 Export fields

The menu *Options|Configuration|Export* allows you to set which fields are exported.

The screenshot shows the 'Export' configuration window. On the left, under the 'Indices' section, there are five options, each with a checked checkbox:

- Indices
- Lowerbound
- Level/Value
- Upperbound
- Marginal

On the right, a text box contains the following instructions:

Here you can specify which fields are to be exported. Indices are the index names. They are absent for a scalar. Sets only export indices. The fields lowerbound, upperbound and marginal are only available for variables and equations.

The following table gives the possibilities for exports:

	set	scalar	parameter	variable	equation
Indices	+		+	+	+
Lower bound				+	+
Level/Value		+	+	+	+
Upper bound				+	+
Marginal				+	+

## 20 Special Values

GAMS data can assume so-called special values: -INF, +INF, EPS, NA, and UNDF. The meaning of these special values is as follows:



-INF	Minus infinity. Mostly used for non-binding lowerbounds.
+INF	Plus infinity. Mostly used for non-binding upperbounds.
EPS	Mostly used for marginals where it can indicate non-basic but numerically zero.
NA	Not available. Not often used.
UNDF	Undefined. Not often used.

When exporting GAMS identifiers we need to map such values to strings that the receiving program can understand. E.g. we could map  $-\text{INF}$  to  $-1.0\text{e}10$  and  $+\text{INF}$  to  $+1.0\text{e}10$ . A good choice for EPS would be 0.0.

The mapping can be specified in *Options|Configuration|Special Values*:

Special values such as INF and EPS may need to be exported as numbers instead of symbols. In such a case you may want to specify  $\text{INF} \rightarrow 1.0\text{e}8$ ,  $\text{EPS} \rightarrow 0.0$ , etc. This mapping is ignored when exporting to a GAMS include file.


INF:

-INF:

EPS:

NA:

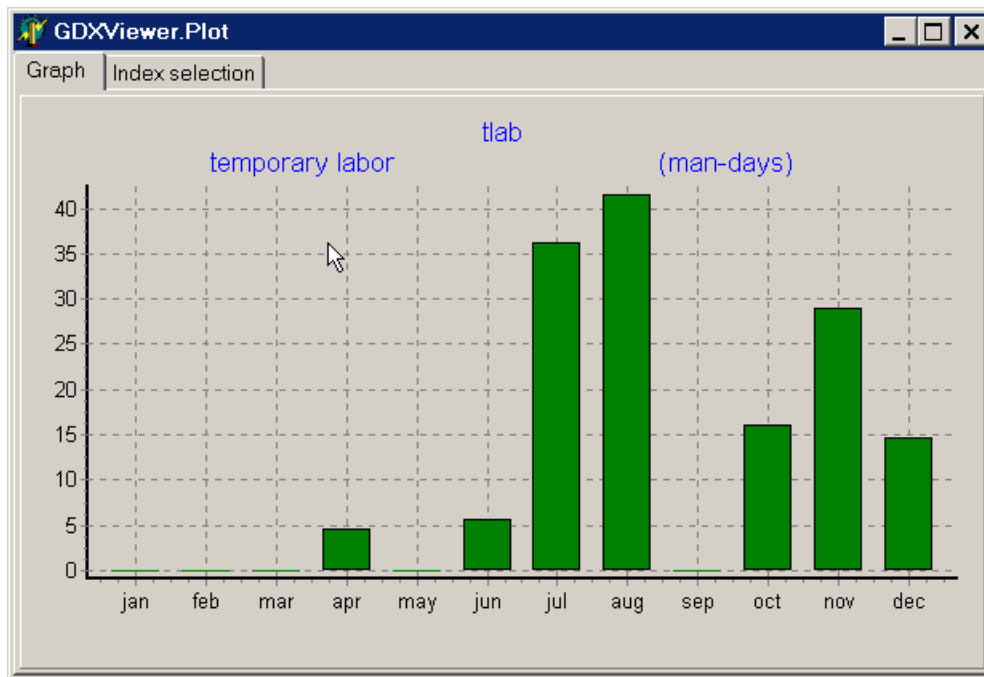
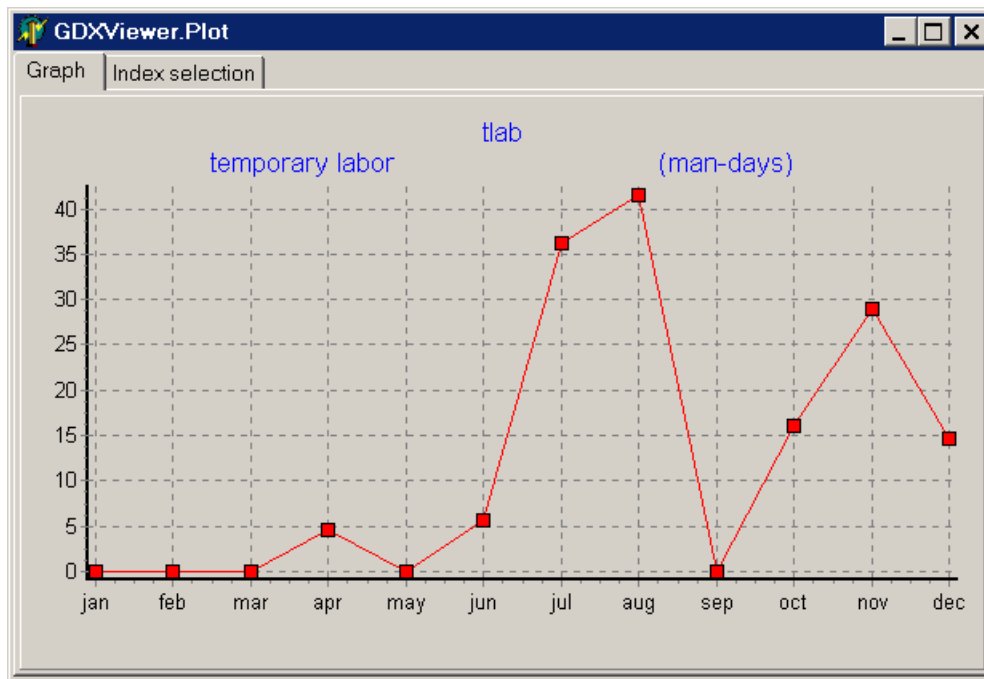
UNDF:

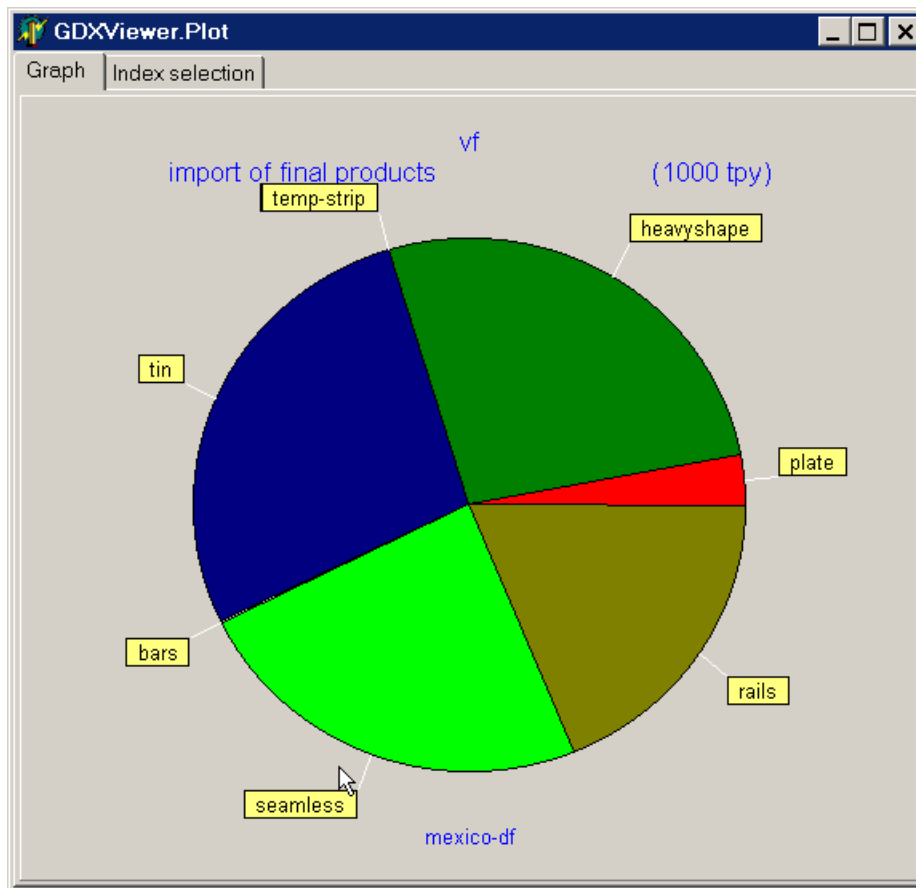
 Defaults

When we export to a GAMS include file all special values are understood, so the mapping is not used. The defaults button will reset the mapping to their default values.

## 21 Plotting data

GDXVIEWER has a built-in facility to quickly plot data. It includes LINE, BAR and PIE charts, examples are shown below. The plots can be made through the menu *File|Plot*.





For multi-dimensional data it may be needed to take a “slice” of the data to make meaningful graphs. In the example above we plotted a two dimensional quantity **vf** which looks like:

GDXViewer v2.7

File Options About

vf (import of final products) (1000 tpy))

Variables	dim1	dim2	lo	level	up	marginal
zm	plate	mexico-df	0	7.199999999...	INF	0
zr	plate	puebla	0	2.1	INF	0
zs	plate	queretaro	0	0	INF	0.109629999...
xm	plate	san-luis	0	0	INF	0.11932
xr	plate	monterrey	0	47.25	INF	0.20501
xs	plate	guadalajara	0	1.05	INF	0
xf	plate	l-cardenas	0	1.05	INF	0
ur	plate	coatzacoal	0	1.05	INF	0
us	heavyshape	mexico-df	0	62.22	INF	0
e	heavyshape	puebla	0	3.74	INF	0
vs	heavyshape	queretaro	0	0	INF	0.109519430...
vf	heavyshape	san-luis	0	0	INF	0.119209430...
cost	heavyshape	monterrey	0	0	INF	0.204899430...
recurrent	heavyshape	guadalajara	0	72.42	INF	0
transport	heavyshape	l-cardenas	0	2.38	INF	0
import	heavyshape	coatzacoal	0	0.51	INF	0
export	lightshape	mexico-df	0	0	INF	0.427469999...
	lightshape	puebla	0	0	INF	0.498389999...
	lightshape	queretaro	0	0	INF	0.493969999...
	lightshape	san-luis	0	0	INF	0.419869999...
	lightshape	monterrey	0	0	INF	0.352799999...
	lightshape	guadalajara	0	0	INF	0.424999999...
	lightshape	l-cardenas	0	0	INF	0.501759999...
	lightshape	coatzacoal	0	0	INF	0.191049999...
	rebars-l	mexico-df	0	0	INF	0.002469999...
	rebars-l	puebla	0	0	INF	0.076149999...
	rebars-l	queretaro	0	0	INF	0.068969999...
	rebars-l	san-luis	0	0	INF	0.01159
	rebars-l	monterrey	0	0	INF	0.097279999...
	rebars-l	guadalajara	0	0.324345491...	INF	0
	rebars-l	l-cardenas	0	0	INF	0.076759999...

Input file: E:\models\mexls.gdx 100%

In this case we want to plot a pie graph of **vf(\*,'mexico-df')** which can be specified in the index-selection tab:

dim1: **PLOTTED**

dim2: **mexico-df**

dim3:

dim4:

dim5:

dim6:

dim7:

dim8:

dim9:

dim10:

Plot Type

☐ Line

☐ Bar

☒ Pie

Here you can set which dimensions are held constant and which ones are varied. To fix a dimension choose a set element. Choose **PLOTTED** if it can be varied. Usually only one dimension varies.

## 22 Cube view

GDXVIEWER has a Cube View which allows to select rows and columns in a flexible way. In the example below we show a six dimensional variable where three dimensions are fixed, one dimension is chosen for the rows and two dimensions are chosen for the columns.

CubeForm — land (land occupation by month)

File

dim1 ⌵ COLUMN ⌵

dim2 ⌵ pmw ⌵

dim3 ⌵ bullock ⌵

dim4 ⌵ COLUMN ⌵

dim5 ⌵ standard ⌵

dim6 ⌵ ROW ⌵

dim7 ⌵

dim8 ⌵

dim9 ⌵

dim10 ⌵

☒ OK

	cotton	rab-fod	gram	mus+rap	kha-fod
	standard	standard	standard	standard	standard
jan	1	1	1	1	0
feb	1	1	1	1	0
mar	1	1	1	1	0.5
apr	1	1	0	0	0.5
may	1	1	0	0	0.5
jun	1	0	0	0	1
jul	0.5	0	0	0	1
aug	0	0	0	0	1
sep	0	0	0.5	0	0.5
oct	0	1	1	1	0.5
nov	0	1	1	1	0
dec	0	1	1	1	0

100%

Below are some of the possibilities using parameter **d(i,j)** from the **trnsport.gms** model:

dim1 <small>⌵</small> ROW <small>⌵</small>		new-york	chicago	topeka
dim2 <small>⌵</small> COLUMN <small>⌵</small>	seattle	2.5	1.7	1.8
	san-diego	2.5	1.8	1.4

dim1 <small>⌵</small> ROW <small>⌵</small>	seattle	new-york	2.5
dim2 <small>⌵</small> ROW <small>⌵</small>	seattle	chicago	1.7
	seattle	topeka	1.8
	san-diego	new-york	2.5
	san-diego	chicago	1.8
	san-diego	topeka	1.4

dim1 <small>⌵</small> COLUMN <small>⌵</small>	seattle	seattle	seattle	san-diego	san-diego	san-diego
dim2 <small>⌵</small> COLUMN <small>⌵</small>	new-york	chicago	topeka	new-york	chicago	topeka
	2.5	1.7	1.8	2.5	1.8	1.4

dim1	<small>Make</small> COLUMN <small>Make</small>		seattle	san-diego
dim2	<small>Make</small> ROW <small>Make</small>		new-york	2.5
			chicago	1.7
			topeka	1.8
				1.4
dim1	seattle		new-york	2.5
dim2	<small>Make</small> ROW <small>Make</small>		chicago	1.7
			topeka	1.8
dim1	seattle		new-york	chicago
dim2	<small>Make</small> COLUMN <small>Make</small>		2.5	1.7
				1.8
dim1	seattle		2.5	
dim2	new-york			

## 23 Exporting cubes

After creating a cube view, we can export that configuration by a right mouse click:

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4

Export	Text File
Transpose	Aligned Text File
	CSV File
	Excel .XLS File
	Access .MDB File
	HTML File
	XML File

Exporting a cube will only export the selected slice (if certain dimensions are held fixed) and depending on the target format it will preserve the layout, e.g. an exported aligned text file can look like:

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4

Similarly, the XLS file can look like:

xxx.XLS					
	A	B	C	D	E
1		new-york	chicago	topeka	
2	seattle	2.5	1.7	1.8	
3	san-diego	2.5	1.8	1.4	
4					
5					

## 24 Commandline operation

The GDXViewer utility from version 2.3 accepts several command line parameters, so it can be used in a batch environment. When running in batch mode, the same configuration and option settings are used as for the interactive system and they can be changed by running GDXviewer interactively using the Options menu (the settings are saved in an INI file). It is advised to first run the program interactively until the results are as intended.

### Single parameter

A single parameter is the filename of the GDX file. GDXViewer will load this file, and will continue to run interactively.

Example: .

```
Gdxviewer.exe test.gdx
```

### XLS writing

To write an XLS file, one can use the syntax *i=inputfile.gdx xls=outputfile.xls id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx xls=d:\tmp\result.xls id=x';
```

### Text file writing

To write a text file, one can use the syntax *i=inputfile.gdx txt=outputfile.txt id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx txt=d:\tmp\result.txt id=x';
```

### CSV file writing

To write a CSV file, one can use the syntax *i=inputfile.gdx csv=outputfile.csv id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx csv=d:\tmp\result.csv id=x';
```

### HTML file writing

To write an HTML file, one can use the syntax *i=inputfile.gdx html=outputfile.html id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx html=d:\tmp\result.html id=x';
```

### XML file writing

To write an XML file, one can use the syntax *i=inputfile.gdx xml=outputfile.xml id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx xml=d:\tmp\result.xml id=x';
```

### GAMS include file writing

To write a GAMS include file, one can use the syntax *i=inputfile.gdx inc=outputfile.inc id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx inc=d:\tmp\result.inc id=x';
```

#### Access MDB file writing

To write a Access MDB file, one can use the syntax *i=inputfile.gdx mdb=outputfile.mdb id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx mdb=d:\tmp\result.mdb id=x';
```

#### Excel Pivot Table writing

To write a file containing a pivot table, one can use the syntax *i=inputfile.gdx pivot=outputfile.xls id=x*. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

#### SQL Database Table writing

To write a table to an SQL database, first interactively configure the connection to the database. The Export SQL Database option allows you to see if a connection succeeded and if the correct database was accessed. The configuration information is written to the SQLVIEWER.INI configuration file. The information in this file is used also when performing a batch command-line operation.

The syntax is: *i=inputfile.gdx sql id=x*.

A complete example is:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx sql id=x';
```

If you need to access several different databases, you can copy the file SQLVIEWER.INI (located in the directory where SQLVIEWER.EXE is placed). To tell GDXVIEWER to read a different INI file, you can say:

```
execute_unload 'd:\tmp\result.gdx', x;  
execute 'gdxviewer.exe i=d:\tmp\result.gdx ini=copy.ini sql id=x';
```

GDXVIEWER uses the MS Access and MS Excel applications as COM Object to write files in XLS (both XLS and PIVOT commands) or MDB format. Those applications may write to C:\My Documents in case no full path is specified. Other formats use the default GAMS working directory. In case when running under the IDE this is the location of the project file (\*.GPR).

If a path or file name contains a blank, then it is possible to surround the name by double quotes as in:

```
execute_unload 'result.gdx', x;  
execute 'gdxviewer.exe i=result.gdx csv="c:\my documents\result.csv" id=x';
```

Under windows 98 ME the call

```
execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

---



will cause GAMS to continue while GDXVIEWER is executing. If we use:

```
execute '=gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

GAMS will wait until **gdxviewer.exe** is terminated before executing more statements. This situation is different under other operating systems such as XP and NT.

## 25 Notes

### GDX

**GDX** stands for **G**ams **D**ata **E**xchange. It is a binary file format to get data in and out of GAMS.