GAUSS™ Data Tool

Version 8.0

Aptech Systems, Inc.

Documentation Revision: 832

Part Number: 005629

Contents

Chapter 1

Getting Started

1.1 Overview

GAUSS Data Tool (**GDT**) is a standalone program for working with **GAUSS** data sets. **GDT** loads columns of a data set into a workspace as vectors where they can be transformed or modified using simple intuitive statements. A variety of simulation models can create new data sets and, using single or multiple imputations, missing data can be included in new versions of data sets.

1.2 GAUSS Data Tool Basics

Within the **GDT** session, **GDT** creates a workspace for each open data set and then loads that workspace by placing each data set variable in a separate $N \times 1$ column vector.

For a list of available **GDT** commands, enter:

```
help
```

For help on a specific command, enter:

```
help command_name
```

In the Windows GUI, to display the dialog interface for a command, type:

command_name -

For example:

```
sort -
```

To list the data sets in the current directory, enter:

```
ld
```

To change directories, enter:

`cd` *directory*

To open a data set, enter:

`open` *file_name*

where *file_name* is the name of the file containing the desired data set in the current directory.

Or

`open` *file_path*

where *file_path* is the path of the desired data set.

Example:

```
open freqdata
open /data/freqdata
```

Each open data set has a *handle*, which is used to refer to the data set in many **GDT** commands. Most **GDT** commands work on the *active* data set. A data set becomes the active data set when it is opened or specified in the **use** command.

To make an open data set the active data set, enter:

`use` *handle*

where *handle* is the handle of an open data set.

To see the list of open data sets, enter:

```
open
```

In the resulting list, the *handle* is shown at the left while the active data set is indicated with a right angle bracket (>) . An asterisk (*****) indicates the data has been modified since the data set was opened or committed.

For a list of variables in the active data set, enter:

```
lv
```

For descriptive statistics on the active data set, enter:

```
stats
```

Transformations on the variables are possible with simple assignment statements.

Example: The following statement modifies the active data set variable `x1` by changing each element to its exponential.

```
x1 = exp(x1)
```

To save active data set changes to the disk data set file, enter:

```
commit
```

To discard active data set changes without saving them to the disk data set file, enter:

```
rollback
```

1.3 User Interfaces

1.3.1 Windows Graphical User Interface

Screen Layout

Within the Windows **GDT** session, program output is displayed in the screen area above the **GDT** prompt. This area is read-only and keyboard entries here will automatically move the cursor to the **GDT** prompt and begin input. However, text in the output area can be copied and pasted to the input area (the **GDT** prompt line and any lines below it) or to other Windows applications.

Basic Input Process

GDT commands and statements are entered at the **GDT** prompt. Pressing [Enter] after entering a command or statement causes that statement or command to be executed.

Each executed command and statement of the **GDT** session is recorded in the command history. To retrieve a previously issued command or statement, use [Ctrl]+[Up Arrow] and [Ctrl]+[Down Arrow] to scroll backward and forward through the command history. The command history only contains commands and statements from the current **GDT** session.

The following are graphical interface commands:

cls	Clears the entire screen.
cd	Changes the current directory; if no arguments are included, the current directory is set to the user's home directory.
dir	Display directory listing–similar to the DOS 'dir' command.
dos, shell	Spawns a DOS shell.
explore	Invoke the Windows Explorer file browser application.
[Ctrl]+[Up Arrow]	Scrolls backward through the command history to reuse a previously issued command.
[Ctrl]+[Down Arrow]	Scrolls forward through the command history to reuse a previously issued command.

Long lines can be extended by adding a \ to the end of a line.

Multi-line Input

The syntax for several **GDT** statements requires multiple line entries.

To enter multi-line mode, do one of the following steps at the **GDT** prompt.

Press [Ctrl]+[Enter].

Or

Enter a backslash (\).

Result: **GDT** enters multi-line mode by creating an input line with a right angle bracket (>) at the left.

After each line entry, press [Enter].

Result: **GDT** adds a new input line.

To edit a line entry, move the cursor to the desired line then modify the line content.

When all line entries have been entered and desired changes made, exit multi-line mode by doing one of the following steps:

Press [Esc].

Result: **GDT** exits multi-line mode without saving the line entries to the command history or executing them.

Or

Press [Enter] on the new empty line.

Result: **GDT** saves line entries to the command history then exits multi-line mode without executing the lines.

Or

Enter a forward slash (/) on the new empty line.

Result: **GDT** saves the line entries to the command history, exits multi-line mode, and executes the lines.

Note: The forward slash (/) does not become a line entry.

1.3.2 Terminal Interface

Screen Layout

Within the Terminal **GDT** session, program output is displayed in the screen area above the **GDT** prompt. Since this is a read-only area, the cursor can only be placed within the **GDT** prompt input area. However, text in the output area can be copied and pasted to the **GDT** prompt input area or other Terminal applications.

Basic Input Process

GDT commands and statements are entered at the **GDT** prompt. Pressing [Enter] after entering a command or statement causes that statement or command to be executed.

Long lines can be extended by adding a \ to the end of a line.

Multi-line Input

The syntax for several **GDT** statements requires multiple line entries.

Several line-editing commands reference the current line. The current line is the last line in the buffer that was accessed.

To enter multi-line mode:

Enter a backslash (\) all by itself at the **GDT** prompt.

Result: **GDT** enters multi-line mode by creating an input line with a line number at the left; this line number is used when referencing a particular line in other line-editing commands.

After each line entry, press [Enter].

Result: **GDT** adds the line entry to the buffer and creates a new input line with an incremented line number.

When all line entries have been entered, exit multi-line mode by doing one of the following steps.

Press [Enter] at the new input line.

Result: **GDT** exits multi-line mode without executing the buffer content.

Or

Enter a forward slash (/) on the new empty line.

Result: **GDT** exits multi-line mode and executes the buffer content.

Note: The forward slash does not become a line entry.

Buffer content remains available for execution or editing until multi-line mode is entered again or the **clear** line-editing command is entered.

Edit the buffer content by entering the following commands in regular (non-multi-line) mode.

a	Appends text to the end of the current line.
del [*n*]	Deletes the current line or line number *n*.

clear	Deletes all lines from the line buffer.
ia	Inserts a line after the current line and makes the new line the current line.
ib	Inserts a line before the current line and makes the new line the current line.
l	Lists all line entries in the buffer; the asterisk (*) in the list indicates which line is the current line.
line no.	Lists the specified line number and makes it the current line.
/	Executes previous buffer contents.
\	Clears previous buffer contents and enters multi-line mode.

The following non-line-editing commands are entered in multi-line mode.

/	Exits multi-line mode and executes the buffer contents; does not become a line entry.
[Enter]	Exits multi-line mode without executing the buffer content; buffer content remains available for execution.

Session Example

Windows Graphical User Interface (GUI):

```
( gdt . ) \
> model linear
> file test
> depvar Y1
> indvar X1,X2
> beta .5,.5
> open
> /

( gdt test . )
```

Terminal Interface

Simple multiple line entry.

```
( gdt ) \
1 model linear
2 file test
3 depvar Y1
4 indvar X1,X2
5 beta .5,.5
6 open
7 /
```

Line entry and editing

```
( gdt test . )


( gdt . ) \
   1  This is line one.
   2  This is line two.
   3  This is line three
   4
( gdt . ) l
   1  This is line one.
   2  This is line two.
   3* This is line three
( gdt . ) a this is appended to line three
   3* This is line threethis is appended to line three
( gdt . ) l
   1  This is line one.
   2  This is line two.
   3* This is line threethis is appended to line three
( gdt . ) 1
   1* This is line one.
( gdt . ) l
   1* This is line one.
   2  This is line two.
   3  This is line threethis is appended to line three
( gdt . ) del 3
( gdt . ) l
   1* This is line one.
   2  This is line two.
( gdt . ) ib This is the new line one
( gdt . ) l
   1* This is the new line one
   2  This is line one.
   3  This is line two.
( gdt . )
```

1.4 Scalar and Element by Element Operations

GAUSS has two versions of certain operators:

> **Scalar-returning** operators, like >=, return a single 1 or 0 (true or false) result depending on whether every element in the left operand matches the comparison requirements with the corresponding elements in the right operand.

Example: `z = x >= y`

Result:

z will be set to 1 if every element in x is >= the corresponding element in y.

z will be set to 0 if any element in x is not >= the corresponding element in y.

Element by element operators, like **.>=**, return a matrix whose values are 1 or 0 (true or false) depending on whether the elements in the left operand match the comparison requirements with the corresponding elements in the right operand.

Example: `z = x .>= y`

Result: Matrix z is created as follows:

For every element in x that is >= the corresponding y element, the corresponding z element is set to 1.

For every element in x that is not >= the corresponding y element, the corresponding z element is set to 0.

The **allops** and **dotops** commands toggle between non-dot (scalar) and dot (element by element) interpretations of these operators in transformation expressions. This affects comparison and logical operators as well as * and /.

Example:

With **allops** in effect, `x1 >= x2` will return a scalar result and `x1 .>= x2` will return an element by element result.

With **dotops** in effect, `x1 >= x2` will behave the same as `x1 .>= x2`, which returns an element by element result.

Code inside procedures is not affected by this setting.

For a more detailed discussion of element by element operations, refer to the *Element by Element Operators* section of the **GAUSS Users Guide**.

1.5 Simulating Data

The **model** command is used to generate simulated data sets. The syntax for a default simulation is:

model *model_name*

where *model_name* is the name of an available model.

For a complete list of available model names for your **GDT** version, enter:

```
help model
```

For help about a specific model, enter:

```
help model model_name
```

Each model type has a variety of options that are available for generating simulated data with specific properties. These options are specified using multi-line mode, described in *User Interfaces* (Section1.3).

Example:

```
( gdt ) \
1 model arima
2 file arsim
3 depvar returns
4 indvar x1,x2
5 vcx 1, .4, 1
6 ar .3, .1
7 ma .1
8 open
9 /
```

ARIMA

The underlying model is as follows:

$$\Phi(L)(1-L)^d y_t = X\beta + \Theta(L)\epsilon_t$$

where L is the lag operator, and

$\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \ldots - \phi_p L^p$

$\Theta(L) = 1 + \theta_1 L + \theta_2 L^2 + \ldots + \theta_q L^q$

and d is the level of integration

GARCH

The underlying model is as follows:

$$\epsilon_t = y_t - x_t\beta - \delta\sigma_t$$

where δ is an "in mean" coefficient

Define

$$\epsilon_t \equiv \eta_t\sigma_t$$

where $E(\eta_t) = 0$, $Var(\eta_t) = 1$, and

$$\sigma_t^2 = \omega + \alpha_1\sigma_{t-1}^2 \ldots \alpha_p\sigma_{t-p}^2 + \epsilon_t + (\beta_1 + \tau_1\zeta(\epsilon_{t-1}))\epsilon_{t-1} \ldots + (\beta_q + \tau_q\zeta(\epsilon_{t-q}))\epsilon_{t-q}$$

where τ_t are "asymmetry" coefficients and

$$\zeta(\epsilon_t) = \begin{cases} 1 & : \quad \epsilon_t < 0 \\ 0 & : \quad \epsilon_t >= 0 \end{cases}$$

LINEAR

The underlying model is as follows:

$$y = x\beta + \epsilon$$

where y is $N \times L$, x is $N \times K$, β is $K \times L$, ϵ is $N \times L$

LOGIT

Binary outcomes are generated in accordance with:

$$Pr(y = k|x) = \frac{e^{k\mu}}{1 + e^{\mu}}$$

where $k = 0, 1$, and $\mu = x\beta$

If a random error term is selected then $\mu = x\beta + \epsilon$.

ORDLOGIT

Ordered outcomes are generated in accordance with:

$$Pr(y = m|x) = \Lambda(\tau_m - \mu) - \Lambda(\tau_{m-1} - \mu)$$

where $\mu = x\beta$, and

$$\Lambda(z) = \frac{e^z}{1 + e^z}$$

If a random error term is selected then $\mu = x\beta + \epsilon$.

ORDPROBIT

Ordered outcomes are generated in accordance with:

$$Pr(y = m|x) = \Phi(\tau_m - \mu) - \Phi(\tau_{m-1} - \mu)$$

where $\mu = x\beta$, and

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{\infty}^{z} e^{-\frac{t^2}{2}} dt$$

If a random error term is selected then $\mu = x\beta + \epsilon$.

PROBIT

Binary outcomes are generated in accordance with:

$$Pr(y = 1|x) = \frac{1}{\sqrt{2\pi}} \int_{\infty}^{\mu} e^{-\frac{t^2}{2}} dt$$

where $Pr(y = 0|x) = 1 - Pr(y = 1|x)$, and $\mu = x\beta$.

If a random error term is selected then $\mu = x\beta + \epsilon$.

Chapter 2

Import

2.1 Overview

import is a multi-line **GDT** command that converts data files into **GAUSS** data sets. It currently supports conversion from both ASCII and Excel files.

2.2 ASCII file conversion

Import may be used to convert both delimited and packed ASCII files. To execute, enter

```
import ascii
```

as the first line of a multi-line statement. Each succeeding line should contain a subcommand.

2.2.1 Command Summary

The following subcommands are supported for importing ASCII files:

append Append data to an existing file.

cmdfile The name of an **import** command file to run.

complex	Treat data as complex variables.
dateformat	Specify date format. Can be overridden on per variable basis.
delimit	Specify delimiter in ASCII input file.
input	The name of the ASCII input file.
invar	Input file variables (column names).
invarline	Get input file variables from specified line in input file.
msym	Specify missing value character.
nocheck	Do not check data type or record length.
open	Open data set after conversion.
output	The name of the **GAUSS** data set to be created.
outtyp	Output data type.
outvar	List of variables to be included in output file.
skip	Skip specified number of lines from beginning of input file.
typecase	Change case of variable names in output file to reflect data type.
vartype	Data types of input file variables.
vartypeline	Get data types of input file variables from specified line in input file.

The principle commands for converting an ASCII file that is delimited with spaces, commas, or tabs are given in the following example:

```
( gdt ) \
1 import ASCII
2 input agex.asc
3 output agex
4 invar $ Race # Age Pay %('MO/DD/YYYY') Hiredate $ Sex Region
5 outvar Region Age Sex Pay
6 outtyp d
7 /
```

From this example, a soft delimited ASCII file `agex.asc` is converted to a double precision **GAUSS** data set `agex.dat`. The input file has six variables, and thus it will be interpreted as having six columns:

column	name	data type
1	Race	character
2	Age	numeric
3	Pay	numeric
4	Hiredate	date
5	Sex	character
6	Region	character

The output file will have 4 columns since the first and the fourth columns of the input file (Race and Hiredate) are not included in the output variables. The columns of the output file are:

column	name	data type
1	Region	character
2	Age	numeric
3	Sex	character
4	Pay	numeric

The variable names are saved in the file header. By default, their case will be preserved.

The $ in the **invar** statement specifies that the variables that follow are character type. The # specifies numeric, and the % specifies date. If a % is not followed by a date format string, then the default will be used. The default date format string is 'YYYY-MO-DD'. See vartype for more information on valid formats for date variables. If no variable type is specified in an **invar** statement, the default is numeric.

Comments in **import** command files begin with // and continue to the end of the line.

2.2.2 Commands

A detailed explanation of each subcommand follows.

- **append**

Instructs **import** to append the converted ASCII data to an existing data set:

```
append
```

No assumptions are made regarding the format of the existing file. You should make certain that the number, order and type of data converted match the existing file. **import** creates v96 format data files, so it will only append to v96 format data files.

- **cmdfile**

Specifies the name of an **import** command file, which contains the subcommands for **import**. The full path name can be used in the file specification.

For example, the command:

```
cmdfile data.cmd
```

will expect an ASCII command file in the current working directory or the **GDT** source path.

For Windows the command:

```
output \research\data\myfile.cmd
```

creates the file `myfile.cmd` on the `\research\data` directory, or on UNIX,

```
cmdfile /research/data/myfile.cmd
```

specifies a file to be located in the `/research/data` directory.

You may use **cmdfile** in a **GDT** session as follows:

```
( gdt ) \
1 import ASCII
2 cmdfile myfile.cmd
3 /
```

In this example, **import** will find and run the `myfile.cmd` file in the current working directory. The command file should contain all of the **import** subcommands needed for the translation, with a semicolon on the end of each subcommand. **import** command files support a superset of the commands available in an **ATOG** command file, and **GDT** will execute **ATOG** command files unchanged with the exception that the default type for the output data set is double precision.

For example, `myfile.cmd` could read as follows:

```
input day1.asc;
output day1;
invarline 1;
vartype # # $ # $;
skip 1;
```

- **complex**

Instructs **import** to convert the ASCII file into a complex **GAUSS** data set:

```
complex
```

Complex **GAUSS** data sets are stored by rows, with the real and imaginary parts interleaved, element by element. **import** assumes the same structure for the ASCII input file, and will thus read TWO numbers out for EACH variable specified.

complex cannot be used with packed ASCII files.

• dateformat

Specifies the format string to be used for inputting date data within the current **import** job. **dateformat** may be overridden on a per variable basis by including a date format string after the % type specifier for that variable. If no date format string is specified with either **dateformat** or **vartype**, then the default will be used. The default date format string is 'YYYY-MO-DD'.

This example:

```
dateformat 'MO/DD/YYYY'
invar $ Name # Age % Bdate %('HH:MI:SS') Btime
```

indicates that the format string 'MO/DD/YYYY' should be used to input the `Bdate` variable, while 'HH:MI:SS' should be used to input `Btime`.

A date format string must contain a single delimiter between each element and be enclosed by single quotes. The following date elements are supported:

YYYY	4 digit year
YR	Last two digits of year
MO	Number of month, 1-12
DD	Day of month, 1-31
HH	Hour of day, 0-23
MI	Minute of hour, 00-59
SS	Second of minute, 00-59

If `YR` is specified and the two-digit year is less than or equal to 37, then 2000 will be added to it. Otherwise, 1900 will be added.

• delimit

Specifies the delimiter in the ASCII input file. The format is as follows:

```
delimit (d, N)
```

or

```
delimit (d)
```

where *d* is the delimiter. The second parameter is the letter **N**. If present, it indicates that **import** should expect to find the same number of delimiters as elements (variables

* observations) in the ASCII file. If the parameter is not present, **import** will expect no delimiter after the last element in the file. See the "Hard Delimited ASCII File" section under the **invar** command for more information.

- **input**

Specifies the file name of the ASCII file to be converted. The full path name can be used in the file specification.

For example, the command:

```
input data.raw
```

will expect an ASCII data file in the current working directory.

In Windows the command:

```
input \research\data\myfile.asc
```

specifies a file to be located in the `\research\data` subdirectory, whereas on UNIX

```
input /research/data/myfile.asc
```

specifies a file to be located in the `/research/data` directory.

- **invar**

Soft Delimited ASCII Files

Soft delimited files may have spaces, commas, tabs, or linefeeds as delimiters between elements. Two or more consecutive delimiters with no data between them are treated as one delimiter.

```
invar Age $ Name Sex # Pay %('MO/DD/YYYY') Hdate Var[1:10] X[005]
```

The **invar** command above specifies the following variables:

column	name	data type
1	Age	numeric
2	Name	character
3	Sex	character
4	Pay	numeric
5	Hdate	date
6	Var01	numeric
7	Var02	numeric
8	Var03	numeric
9	Var04	numeric
10	Var05	numeric
11	Var06	numeric
12	Var07	numeric
13	Var08	numeric
14	Var09	numeric
15	Var10	numeric
16	X001	numeric
17	X002	numeric
18	X003	numeric
19	X004	numeric
20	X005	numeric

As the input file is translated, the first 20 elements will be interpreted as the first row (observation), the next 20 will be interpreted as the second row, and so on. If the number of elements in the file is not evenly divisible by 20, the final incomplete row will be dropped and a warning message will be given.

Hard Delimited ASCII Files

Hard delimited files have a printable character as a delimiter between elements. Two delimiters without intervening data between them will be interpreted as a missing. If **\n** is specified as a delimiter, the file should have one element per line and blank lines will be considered missings. Otherwise, delimiters must be printable characters. The dot '.' is illegal and will always be interpreted as a missing value. To specify the backslash as a delimiter, use \\. If **\r** is specified as a delimiter, the file will be assumed to contain one case or record per line with commas between elements and no comma at the end of the line.

For hard delimited files the **delimit** subcommand is used with the **invar** command. The **delimit** subcommand has two optional parameters. The first parameter is the delimiter; the default is a comma. The second parameter is the letter **N**. If present, it indicates that **import** should expect to find the same number of delimiters as elements (variables * observations) in the ASCII file. If the parameter is not present, **import** will expect no delimiter after the last element in the file.

This example:

```
invar delimit(, N) $ name # var[5]
```

will expect a file like this:

```
BILL ,  222.3,  123.2,  456.4,  345.2,  533.2,
STEVE,  624.3,  340.3,       ,  624.3,  639.5,
TOM  ,  244.2,  834.3,  602.3,  333.4,  822.5,
```

This example:

```
invar delimit(,) $ name # var[5]
```

or

```
invar delimit $ name # var[5]
```

will expect a file like this:

```
BILL ,  222.3,  123.2,  456.4,  345.2,  533.2,
STEVE,  624.3,  340.3,       ,  624.3,  639.5,
TOM  ,  244.2,  834.3,  602.3,  333.4,  822.5
```

The difference between specifying N or N-1 delimiters can be seen here:

```
456.4,  345.2,  533.2,
     ,  624.3,  639.5,
602.3,  333.4,
```

If the **invar** statement had specified 3 variables and N-1 delimiters, this file would be interpreted as having three rows containing a missing in the 2,1 element and the 3,3 element like this:

```
456.4  345.2  533.2
.      624.3  639.5
602.3  333.4  .
```

If N delimiters had been specified, this file would be interpreted as having two rows, and a final incomplete row that is dropped:

```
456.4  345.2  533.2
.      624.3  639.5
```

The spaces were shown only for clarity and are not significant in delimited files so:

```
BILL,222.3,123.2,456.4,345.2,533.2,
STEVE,624.3,340.3,,624.3,639.5,
TOM,244.2,834.3,602.3,333.4,822.5
```

would work just as well. Linefeeds are significant only if **n** is specified as the delimiter, or when using **r**.

This example:

```
invar delimit(\r) $ name # var[5]
```

will expect a file with no comma after the final element in each row:

```
BILL ,  222.3,  123.2,  456.4,  345.2,  533.2
STEVE,  624.3,  340.3,  245.3,  624.3,  639.5
TOM  ,  244.2,  834.3,  602.3,  333.4,  822.5
```

Packed ASCII Files

Packed ASCII files must have fixed length records. The **record** subcommand is used to specify the record length and variables are specified by giving their type, starting position, length, and the position of an implicit decimal point if necessary.

Note that **outvar** is not used with packed ASCII files. Instead, **invar** is used to specify only those variables to be included in the output file.

For packed ASCII files the syntax of the **invar** command is as follows:

`invar record=`*reclen (format) variables (format) variables*

where,

reclen the total record length in bytes, including the final carriage return/linefeed if applicable. Records must be fixed length.

format (*start,length.prec*) where:

- **start** starting position of the field in the record, 1 is the first position. The default is 1.
- **length** length of the field in bytes.
- **prec** optional; a decimal point will be inserted automatically **prec** places in from the RIGHT edge of the field.

If several variables are listed after a format definition, each succeeding field will be assumed to start immediately after the preceding field. If an asterisk is used to specify the starting position, the byte following the last field will be assumed. An asterisk in the length position will leave both length and prec unchanged from the previous settings. This is illegal: (3,8.*).

The type change characters $, # and % are used to specify character, numeric, and date types respectively. If % is indicated, it should be followed by a date format string and then the variable format. See **dateformat** for valid date formats. Since fields in packed ASCII files must have a fixed length, date data must contain two digit months, days and hours or be padded with spaces to the correct field length.

Any data in the record that is not defined in a format is ignored.

The examples below assume a 42-byte record with a carriage return/linefeed occupying the last 2 bytes of each record. The data below can be interpreted in different ways using different **invar** statements:

```
          ABCDEFGHIJ01-23-2003123456789012345678901<CR><LF>
          |         |          |          |          |  |   |
position 1        10         20         30         40 41  42
```

This example:

```
invar record=42 $(1,3) group %('MO-DD-YYYY',11,10) deadline #(*,4.2) x[3]
```

will result in:

variable	value	type
group	ABC	character
deadline	01-23-2003	date
x1	12.34	numeric
x2	56.78	numeric
x3	90.12	numeric

In the **GAUSS** data set, date columns are stored in DT Scalar format. In the above example, the value of **deadline** will be a double containing 20030123000000. See **Date And Time Formats** in the "Language Fundamentals" chapter of the *User Guide* for details on the DT Scalar format.

This example:

```
invar record=42 $(1,8) dept (*,2) id # (21,5) wage (*,2) area
```

will result in:

variable	value	type
dept	ABCDEFGH	character
id	IJ	character
wage	12345	numeric
area	67	numeric

- **invarline**

Gets input file variable names from a specified line in the input file.

For example:

```
invarline 1
```

will get the variable names from the first line of the input file. **invarline** may be used only with delimited ASCII files. In a soft delimited ASCII file, the variable names may be delimited with spaces, commas, or tabs. In a hard delimited ASCII file, the variable names must use the same delimiter as the data. To use **invarline** with a hard delimited ASCII file, call **delimit** to specify the delimiter.

invarline should be used in conjunction with **skip** so that **import** will not attempt to retrieve data from the line in the input file which contains the variable names.

- **msym**

Specifies the character or string in the input file that is to be interpreted as a missing value.

This example:

```
msym &
```

defines the character & as the missing value character.

This example:

```
msym NA
```

specifies that each instance of 'NA' in the input file is to be interpreted as a missing value.

The default '.' (dot) will always be interpreted as a missing value unless it is part of a numeric value.

- **nocheck**

Optional, suppresses automatic checking of packed ASCII record length and output data type. The default is to increase the record length by 2 bytes if the second record in a packed file starts with a linefeed, and any files that have explicitly defined character data will be output in double precision regardless of the type specified.

- **open**

After the conversion, the **GAUSS** data set is opened for use in **Data Tool**:

```
open
```

- **output**

The name of the **GAUSS** data set. A file will be created with the extension `.dat`.

For example, on Windows

```
output \research\data\myfile
```

creates the file `myfile.dat` on the `\research\data` directory, or on UNIX,

```
output /research/data/myfile
```

creates the file `myfile.dat` on the `/research/data` directory.

The command:

```
output myfile
```

creates the file in the current working directory.

- **outtyp**

Selects the numerical accuracy of the output file. Use of this command should be dictated by the accuracy of the input data and storage space limitations.

For example:

```
outtyp D
```

will write a double precision output file.

outtyp may be set to any of the following:

D	double precision (default)
F	single precision
I	integer

The ranges of the different formats are:

bytes	data type	significant digits	range
2	integer	4	$-32768 \leq X \leq 32767$
4	single precision	6–7	$8.43\text{x}10^{-37} \leq \|X\| \leq 3.37\text{x}10^{+38}$
8	double precision	15–16	$4.19\text{x}10^{-307} \leq \|X\| \leq 1.67\text{x}10^{+308}$

If the output type is integer, the input numbers will be truncated to integers. If your data has more than 6 or 7 significant digits, you should specify **outtyp** as double.

Character and date data require **outtyp d**. **import** automatically selects double precision when character or date data are specified unless you have also specified **nocheck**.

The precision of the storage selected does not affect the accuracy of **GAUSS** calculations using the data. **GAUSS** converts all data to double precision when the file is read.

- **outvar**

Selects the variables to be placed in the **GAUSS** data set. The **outvar** command needs only the list of variables to be included in the output data set. They can be in any order. If outvar is not used, all of the input variables are written to the output file.

For example:

```
invar $name #age pay $sex %('YYYY/MO/DD') hiredate #var[1:10] x[005]
outvar sex age hiredate x001 x003 var[1:8]
```

column	name	data type
1	sex	character
2	age	numeric
3	hiredate	date
4	x001	numeric
5	x003	numeric
6	var01	numeric
7	var02	numeric
8	var03	numeric
9	var04	numeric
10	var05	numeric
11	var06	numeric
12	var07	numeric
13	var08	numeric

outvar is not used with packed ASCII files.

- **skip**

Skips down a specified number of lines from the beginning of the input file before retrieving data.

For example:

```
skip 5
```

will skip the first five lines in the input file and begin retrieving data from the sixth line. This command may be used to skip any explanatory notes that are included at the beginning of the ASCII file or to skip lines containing variable names and types. See documentation on the **invarline** and **vartypeline** subcommands for further information on getting variable names and types out of an ASCII file.

- **typecase**

typecase should be used only to create backward compatible data sets which may be used in old programs and applications.

Forces the names of character variables to lower case and the names of numeric variables to uppercase. If **typecase** is specified, the date variable type is not supported. If **typecase** is not specified, the case of each variable name will be preserved.

- **vartype**

Specifies the types of the input variables.

For example:

```
vartype $ # # %('MO/DD/YYYY HH:MI') $
```

will specify the types of the input variables as follows:

column	type
1	character
2	numeric
3	numeric
4	date
5	character

If a date variable is specified in a **vartype** statement, then it should be followed by a date format string unless the default is to be used. The default date format string is 'YYYY-MO-DD'. To reset the default date format string for the current **import** job, use the **dateformat** subcommand. A date format string must contain a single delimiter between each element and be enclosed by single quotes. The following date elements are supported:

YYYY	4 digit year
YR	Last two digits of year
MO	Number of month, 1-12
DD	Day of month, 1-31
HH	Hour of day, 0-23
MI	Minute of hour, 00-59
SS	Second of minute, 00-59

If `YR` is specified and the two-digit year is less than or equal to 37, then 2000 will be added to it. Otherwise, 1900 will be added.

It is possible to set both variable names and variable types using **invar**. However, if the variable names are retrieved from the ASCII file with **invarline**, you need to set the variable types using either **vartype** or **vartypeline**. Use **vartype** to set the variable types within your **import** multi-line statement, or **vartypeline** to retrieve the variable types from a specified line in the ASCII file.

The number of variable names must match the number of variable types.

- **vartypeline**

Retrieves the types of the input variables from a specified line in the input file.

For example:

```
vartypeline 2
```

will retrieve the types of the input variables from the second line of the input ASCII file. In this case, line 2 of the ASCII file should contain a delimited series of type specifiers: # to indicate numeric data, $ to indicate character data, and % for date data. A % should be followed by a date format string unless the default is to be used. The default date format string is 'YYYY-MO-DD'. To reset the default date format string for the current **import** job, use the **dateformat** subcommand. A date format string must contain a single delimiter between each element. See **dateformat** for valid date formats.

vartypeline may be used only with delimited ASCII files. In a soft delimited ASCII file, the variable types may be delimited with spaces, commas, or tabs. In a hard delimited ASCII file, the variable types must use the same delimiter as the data.

vartypeline should be used in conjunction with **skip** so that **import** will not attempt to retrieve data from the line in the input file which contains the variable types.

It is possible to set both variable names and variable types using **invar**. However, if the variable names are retrieved from the ASCII file with **invarline**, you need to set the variable types using either **vartype** or **vartypeline**. Use **vartype** to set the variable types within your **import** multi-line statement, or **vartypeline** to retrieve the variable types from a specified line in the ASCII file. The number of variable types found must match the number of variable names indicated by the **invar** or **invarline** command.

2.2.3 Examples

The first example is a soft delimited ASCII file called `agex1.asc`. The file contains seven columns of ASCII data.

```
Jan 167.3 822.4 6.34E06 yes 84.3 100.4
Feb 165.8 987.3 5.63E06 no 22.4 65.6
Mar 165.3 842.3 7.34E06 yes 65.4 78.3
```

The **import** multi-line command is as follows:

```
( gdt ) \
1 import ASCII
2 input /gauss/agex1.asc
3 output agex1
4 invar $month #temp pres vol $true #var[02]
5 outvar month true temp pres vol
6 /
```

The output data set will contain the following information:

name	month	true	temp	pres	vol
type	char	char	numeric	numeric	numeric
case 1	Jan	yes	167.3	822.4	6.34e+6
case 2	Feb	no	165.8	987.3	5.63e+6
case 3	Mar	yes	165.3	842.3	7.34e+6

The data set defaults to double precision since no **outtyp** command is specified.

The second example is a packed ASCII file `xlod.asc` which contains 32-character records.

```
          AEGDRFCSTy023456789606315678900<CR><LF>
          EDJTAJPSTn123958639980648395610<CR><LF>
          GWDNADMSTy198278456597252344510<CR><LF>
          |         |          |          |  |   |
position 1        10         20         30 31  32
```

The **import** multi-line command is as follows:

```
( gdt ) \
1 import ASCII
2 input /gauss/dat/xlod.asc
3 output xlod1
4 invar record=32 $(1,3) client[2] zone (*,1) reg #(20,5) zip
5 /
```

The output data set will contain the following information:

name	client1	client2	zone	reg	zip
type	char	char	char	char	numeric
case 1	AEG	DRF	CST	y	60631
case 2	EDJ	TAJ	PST	n	98064
case 3	GWD	NAD	MST	y	59725

The data set is double precision.

The third example is a hard delimited ASCII file called `cplx.asc`. The file contains six columns of ASCII data:

```
 cvar1,          cvar2,          cvar3,
 456.4,  345.2,  533.2, -345.5,  524.5,  935.3,
-257.6,  624.3,  639.5,  826.5,  331.4,  376.4,
 602.3, -333.4,  342.1,  816.7, -452.6, -690.8
```

The **import** multi-line command is as follows:

```
( gdt ) \
1 import ASCII
2 input /gauss/cplx.asc
3 output cplx
4 invarline 1
5 skip 1
6 vartype # # #
7 delimit
8 complex
9 /
```

The output data set will contain the following information:

name	cvar1	cvar2	cvar3
type	numeric	numeric	numeric
case 1	456.4 + 345.2i	533.2 - 345.5i	524.5 + 935.3i
case 2	-257.6 + 624.3i	639.5 + 826.5i	331.4 + 376.4i
case 3	602.3 - 333.4i	342.1 + 816.7i	-452.6 - 690.8i

The data set is double precision.

2.3 Excel file conversion

Import may also be used to convert Excel files into **GAUSS** data sets. To execute, enter

```
import excel
```

as the first line of a multi-line statement. Each succeeding line should contain a subcommand.

2.3.1 Command Summary

The following subcommands are supported for importing Excel files:

append	Append Excel data to existing **GAUSS** data set.
datarange	Range of data in the input file, default = a1.
datasheet	Sheet in the input file containing the data.
input	The name of the Excel input file.
namerange	Range of input variable names, if any, in the input file.
names	List of input variable names.
namesheet	Sheet in the input file containing input variable names.
open	Open data set after conversion.
output	The name of the **GAUSS** data set to be created.
overwrite	Overwrite the output **GAUSS** data set.
translate	Translate Excel special characters to specified values in **GAUSS** data set.

2.3.2 Example

This example imports a row vector of names starting in cell "a1" in the Excel file, and a matrix of data with upper left starting element in cell "a3". The names and matrix of data are entered into a **GAUSS** data set with file name `test1.dat`.

```
( gdt ) \
1 import excel
2 input test1.xls
3 namerange a1
4 datarange a3
5 output test1
6 /
```

In the following example, the variable names are stored in a column vector starting in cell "a1" and ending in cell "a6". The data are stored by columns in a block with cell "c1" as the upper left corner and cell "h20" as the lower right corner.

```
( gdt ) \
1 import excel
2 input test2.xls
3 namerange a1:a6
4 datarange c1:h20
5 output test2
6 /
```

In this example, either the variable names do not exist in the Excel file, or the user chooses not to use them. Default names $X1, X2, \ldots$ are given in the resulting **GAUSS** data set.

```
( gdt ) \
1 import excel
2 input test3.xls
4 datarange a3
5 output test3
6 /
```


Excel Dates

Data that are typed as dates in the Excel file will be transformed to dates in DT format in the **GAUSS** data set.

Special Characters

Special characters in the Excel file can be transformed to specified numerical values in the **GAUSS** data set. The special characters are

empty

#N/A

#VALUE!

#DIV/0!

#NAME?

#REF!

#NUM!

#NULL!

#ERR

By default they are all converted to **GAUSS** missing values. If a numeric value is wanted, use the **translate** command, specifying both the special character and the numeric value. For example:

```
translate #N/A = 999
```

translates all instances of #N/A to 999 in the **GAUSS** data set.

2.3.3 Commands

A detailed explanation of each subcommand follows.

- **append**

Instructs **import** to append the Excel data to an existing data file:

```
append
```

No assumptions are made regarding the format of the existing file. You should make certain that the number, order and type of data converted match the existing file. **import** creates v96 format data files, so it will only append to v96 format data files.

- **datarange**

Specifies the cell range of the data in the Excel file:

```
datarange a1:c100
```

If the data is in consecutive cells in columns, only the upper left cell needs to be specified:

```
datarange a1
```

- **datasheet**

Specifies the sheet of the data in the Excel file:

```
datasheet 2
```

The default is 1.

- **input**

Specifies the file name of the Excel file to be converted. The full path name can be used in the file specification.

For example, the command:

```
input data.raw
```

will expect an Excel data file in the current working directory.

The command:

```
input \research\data\myfile.asc
```

specifies a file to be located in the `\research\data` subdirectory.

- **namerange**

Specifies the cell range of the names of the columns of the data in the Excel file, if any:

When the **namerange** option is specified as a single cell, the names are assumed to be distributed in a row with names in consecutive cells. Suppose there are 6 columns of data and thus 6 names in the Excel file. Then the option

```
namerange a1
```

is equivalent to

```
namerange a1:f1
```

If the names are listed in a column rather than a row, then the entire range must be specified. For example:

```
namerange a1:a6
```

If names do not exist in the Excel file nor are specified by the **names** command, default names are given: $X1, X2, \ldots, X_k$.

- **names**

If the Excel file does not contain the names of the columns of the data, they can be specified with this command:

```
names age,sex,pay
```

If names are neither specified with this command nor exist in the Excel file, default names are given: $X1, X2, \ldots, X_k$.

- **namesheet**

Specifies the sheet of the variable names in the Excel file:

```
namesheet 2
```

The default is 1.

- **open**

Opens the **GAUSS** dataset in **Data Tool** after creation:

```
open
```

- **output**

The name of the **GAUSS** data set. A file will be created with the extension `.dat`.

For example:

```
output \research\data\myfile
```

creates the file `myfile.dat` on the `\research\data` directory.

The command:

```
output myfile
```

creates the file in the current working directory.

- **overwrite**

This command is used to specify that the output **GAUSS** data set will replace a current **GAUSS** data set with the same name if it exists:

```
overwrite
```

- **translate**

This command specifies that certain special characters in the Excel file are translated to given values in the **GAUSS** data set. By default these special characters are translated to **GAUSS** missing values.

The following are the special characters:

empty

#N/A

#VALUE!

#DIV/0!

#NAME?

#REF!

#NUM!

#NULL!

#ERR

For example, to translate all instances of `#ERR` to 999, enter

```
translate #ERR = 999
```

A separate **translate** command is required for each translation. All special characters not associated with a **translate** command are translated to **GAUSS** missing values.

Chapter 3

Export

3.1 Overview

export is a multi-line **GDT** command that converts **GAUSS** data sets into data files of other types. It currently supports conversion to both ASCII and Excel files.

3.2 ASCII file conversion

Export may be used to convert an active **GAUSS** data set into a delimited ASCII file. To execute, enter

```
export ASCII
```

as the first line of a multi-line statement. Each succeeding line should contain a subcommand.

3.2.1 Command Summary

The following subcommands are supported for exporting ASCII files:

append	Append to already existing output file.
dateformat	Specify format for date data.

drop	Write all variables except those specified.
finaldelim	Specify delimiter to be written at the end of the file.
keep	Write only specified variables.
noquote	Do not quote character and date data.
obsdelim	Specify delimiter to be written between observations.
output	The name of the ASCII data file to write.
overwrite	Overwrite output file if it already exists.
vardelim	Specify delimiter to be written between variables.
writevarnames	Write variable names to output file.
writevartypes	Write variable types to output file.

3.2.2 Commands

A detailed explanation of each subcommand follows.

- **append**

Instructs **export** to append the converted data to an existing ASCII file:

```
append
```

No assumptions are made regarding the format of the existing file. You should make certain that the number, order and type of data converted match the existing file.

- **dateformat**

Specifies the format in which date data should be written to the output file. The format is as follows:

`dateformat` *default_date_format date_format var_name...*

Each *var_name* that is directly specified will be written in the ASCII file according to the *date_format* immediately preceding. Any date variables in the data set that are not directly specified will be written according to the *default_date_format*.

For example, when exporting a data set containing the following variables:

column	name	data type
1	Pname	character
2	Edate	date
3	Start	date
4	End	date
5	Elapsed	date

the command:

```
dateformat 'HH:MI' 'MO/DD/YYYY' Edate
```

specifies that **Edate** is to be written using the format 'MO/DD/YYYY', and **Start**, **End**, and **Elapsed** using the format 'HH:MI'.

A date format string must contain a single delimiter between each element and must be enclosed by single quotes. The following date elements are supported:

YYYY	4 digit year
YR	Last two digits of year
MO	Number of month, 1-12
DD	Day of month, 1-31
HH	Hour of day, 0-23
MI	Minute of hour, 00-59
SS	Second of minute, 00-59

If no date format is specified with the **dateformat** command, any date data encountered in the data set will be written according to the default for importing and exporting date data. The default date format is 'YYYY-MO-DD'.

- **drop**

Specifies variables that are NOT to be written to the output file. For example, when converting a data set containing three variables, **x1, x2** and **x3**, to an ASCII file, the command:

```
drop x1,x2
```

specifies that only variable **x3** be written to the output file.

- **finaldelim**

Specifies the delimiter to be written after the final observation in the output ASCII file.

For example:

```
finaldelim '|'
```

specifies that the final observation in the output file be followed by a vertical bar.

- **keep**

Specifies variables to be written to the output file. For example, when converting a data set containing three variables, **x1**, **x2** and **x3**, to an ASCII file, the command:

```
keep x2,x3
```

specifies that only variables **x2** and **x3** be written to the ASCII file.

- **noquote**

Instructs **export** to write character and date data to the output ASCII file without quotes:

```
noquote
```

By default, character and date data are enclosed by double quotes in the output file.

- **obsdelim**

Specifies the delimiter to be written between each observation in the output ASCII file.

To specify the delimiter to be written after the final observation in the ASCII file, use **finaldelim**.

For example:

```
obsdelim ','
```

specifies that each observation in the output file except the last be followed by a comma.

- **output**

The name of the output ASCII file.

For example, on Windows

```
output \research\data\myfile.asc
```

creates the file `myfile.asc` on the `\research\data` directory, or on UNIX,

```
output /research/data/myfile.asc
```

creates the file `myfile.asc` on the `/research/data` directory.

The command:

```
output myfile.asc
```

creates the file in the current working directory.

- **overwrite**

Overwrites the output ASCII file if it already exists:

```
overwrite
```

By default, **export** will error out if the specified output file exists.

- **vardelim**

Specifies the delimiter to be written between each variable within an observation in the output ASCII file.

To specify the delimiter to be written between observations, use **obsdelim**.

For example:

```
vardelim ', '
```

specifies that a comma and a space be written between each variable within an observation in the output file.

- **writevarnames**

Writes the names of the variables in the the output file before the data:

```
writevarnames
```

The variable names will be enclosed by double quotes unless **noquote** is specified, and they will be delimited in the same way as the data that follows.

- **writevartypes**

Writes the types of the variables in the output file before the data:

```
writevartypes
```

The variable types will be enclosed by double quotes unless **noquote** is specified, and they will be delimited in the same way as the data that follows. A date type specifier, %, will be followed by the corresponding date format string.

3.2.3 Examples

The first example is a **GAUSS** data set called `einfo.dat`, which has five variables and contains the following data:

varname	Ename	bdate	age	pay	hiredate
type	char	date	numeric	numeric	date
case 1	Sue	19710421000000	33	60000	20010115000000
case 2	Tom	19570905000000	47	90000	19981015000000
case 3	James	19641230000000	40	65000	19950401000000
case 4	John	19740528000000	30	58000	19970701000000

The following **export** multi-line command:

```
( gdt . einfo ) \
1 export ASCII
2 output /gauss/einfo.asc
3 writevarnames
4 writevartypes
5 dateformat 'MO/DD/YR'
6 vardelim ', '
7 obsdelim ,
8 /
```

will produce the output ASCII file `einfo.asc` on the `gauss` directory, which will contain:

```
"Ename", "bdate", "age", "pay", "hiredate",
"$", "%('MO/DD/YR')", "#", "#", "%('MO/DD/YR')",
"Sue", "04/21/71", 33, 60000, "01/15/01",
"Tom", "09/05/57", 47, 90000, "10/15/98",
"James", "12/30/64", 40, 65000, "04/01/95",
"John", "05/28/74", 30, 58000, "07/01/97"
```

The second example converts a **GAUSS** data set called `test1.dat`, which has four variables and contains the following data:

varname	Month	X1	X2	X3
type	char	numeric	numeric	numeric
case 1	JAN	162384	105.32	1546
case 2	FEB	123643	462.15	3628
case 3	MAR	102738	362.9	9361

The following **export** multi-line command:

```
( gdt . test1 ) \
1 export ASCII
2 output /gauss/mdata.asc
3 writevarnames
4 writevartypes
5 drop x2
6 noquote
7 vardelim |
8 /
```

will produce the output ASCII file `mdata.asc` on the `gauss` directory, which will contain:

```
Month|X1|X3
$|#|#
JAN|162384|1546
FEB|123643|3628
MAR|102738|9361
```


3.3 Excel file conversion

Export may be used to convert **GAUSS** data sets into Excel files. To execute, enter

```
export excel
```

as the first line of a multi-line statement. Each succeeding line should contain a subcommand.

3.3.1 Command Summary

The following subcommands are supported for exporting Excel files:

datarange	Range of data in the output file, default = a1.
datasheet	Sheet in the output file for the data.
deletefile	Delete existing output file.
drop	Write all variables except those specified.
keep	Write only specified variables.

namerange	Range of names, if to be written to Excel file.
namesheet	Sheet in the output file of names.
output	The name of the output Excel file.
translate	Translate specified **GAUSS** numbers into Excel special characters.
update	Update existing output file.

Names

If namerange is not set, names will not be written to the Excel file. If namerange is set to a single cell, names will be written row-wise. To write names to a column, beginning and ending cells must be specified, e.g., "A1:A5".

Special Characters

Specified numbers in the **GAUSS** data set can be written as special characters in the Excel data set. The special characters are

empty

#N/A

#VALUE!

#DIV/0!

#NAME?

#REF!

#NUM!

#NULL!

#ERR

By default, **GAUSS** missing values will be written as empty cells.

3.3.2 Example

This example exports the data in a **GAUSS** data set called `test.dat` into the first sheet of an Excel file where the upper left element of the data goes into cell "a3" and the names are entered row-wise starting in cell "a1".

```
( gdt . test ) \
1 export excel
2 output test.xls
3 namerange a1
4 datarange a3
5 /
```

In the following example, the names are left out of the Excel file, and **GAUSS** missing values in the data are translated to a special Excel character:

```
( gdt . test ) \
1 export excel
2 output test.xls
3 datarange a3
4 translate . = #N/A
5 /
```


3.3.3 Commands

A detailed explanation of each subcommand follows.

- **datarange**

Specifies the range for the data in the output Excel file. Only the upper left cell is required:

```
datarange c4
```

The default is cell a1;

- **datasheet**

Specifies the sheet of the data in the Excel file:

```
datasheet 2
```

The default is 1.

- **deletefile**

Deletes the output Excel file if it already exists before exporting:

```
deletefile
```

By default, **export** will error out if the specified output file exists.

- **drop**

Specifies columns in the **GAUSS** data set to be excluded from the output Excel file. For example,

```
drop age, pay
```

- **keep**

Specifies columns in the **GAUSS** data set to be included in the output Excel file.

For example,

```
keep age, pay
```

If neither **keep** nor **drop** commands are entered, all columns are included in the Excel file.

- **namerange**

Specifies the cell range in the Excel file for the names. If there isn't a **namerange** command, the names won't be entered into the Excel file.

If a single cell is specified, the names are entered in consecutive cells row-wise. Thus

```
namerange a1
```

for 3 names is equivalent to

```
namerange a1:c1
```

To enter the names in a column, the entire range must be specified:

```
namerange a1:a3
```

- **namesheet**

Specifies the sheet of the variable names in the Excel file:

```
namesheet 2
```

The default is 1.

- **output**

The name of the output Excel file. A file will be created with the extension `.xls`.

For example

```
output \research\data\myfile
```

creates the file `myfile.xls` on the `\research\data` directory.

The command:

```
output myfile
```

creates the file in the current working directory.

- **translate**

This command is used to translate specified values in the **GAUSS** data set into special characters in the Excel file. By default **GAUSS** missing values are translated into empty cells in the Excel file.

The following are the special characters:

empty

#N/A

#VALUE!

#DIV/0!

#NAME?

#REF!

#NUM!

#NULL!

#ERR

For example, to translate all instances of 999 in the **GAUSS** data set into the special character `#ERR`, enter

```
translate 999 = #ERR
```

A separate **translate** command is required for each translation.

- **update**

Instructs **export** to update an existing Excel file with the specified data translation:

```
update
```

This subcommand allows you to overwrite a part of an existing Excel file or append to an Excel file. No assumptions are made regarding the format of the existing file.

3. *EXPORT*

Chapter 4

Statement Reference

Purpose

Creates a new data set variable.

Format

add ⟦*vtype*⟧ *newvar* = *expression*

add ⟦*vtype*⟧ *newvar1*, *newvar2*, ...

Valid vtypes are:

#	numeric (default)
$	character
%	date

Output

A *nobs* × 1 variable is added to the active data set, where *nobs* is the number of observations in the data set.

Remarks

This is a transformation expression and all operators are dot (element by element) operators unless this feature has been turned off using **allops**.

If no *expression* is present, a vector of zeros is added to the data set, which corresponds to a NULL string for character variables and a zeroed out DT for date variables.

If an expression is given, it must produce a vector of length equal to the number of observations in the active data set.

Code Example

```
add # agecat = age >= 21 and age < 65
add $ sex = "Male" * (nsex == 1) + "Female" * (nsex == 2)
```

The example above creates *agecat* with a value of 1 where *age* >= 21 and *age* < 65, otherwise, *agecat* is created with a value of 0. Then the character variable *sex* is created with a value of "Male" where *nsex* is 1 or a value of "Female" where *nsex* is 2.

Session Example

```
( gdt ) open linear
( gdt linear ) lv
> linear                        3 vars,     100 obs,  /data/linear.dat
  Y                                 numeric
  X1                                numeric
  X2                                numeric
( gdt linear ) add ex2 = exp(x2)
( gdt * linear ) lv
> * linear                        4 vars,     100 obs,  /data/linear.dat
  Y                                 numeric
  X1                                numeric
  X2                                numeric
* ex2                               numeric
```

See also

create

Purpose

Turns off interpretation of scalar-returning operators as element by element operators.

Format

allops

Remarks

Non-dot operators in a transformation expression will be interpreted as scalar-returning operators and dot operators will be interpreted as element by element operators.

Code inside of procedures is not affected by this setting.

For more information, refer to *Scalar and Element by Element Operations* (Section 1.4) and *Element by Element Operators* in the **GAUSS Users Guide**.

Code Example

```
allops
```

Session Example

```
( gdt . ) open tobit
( gdt . tobit ) lv
>   tobit                        4 vars,     100 obs,  /data/example/tobit.dat
  Y                                 numeric
  CNST                              numeric
  X1                                numeric
  X2                                numeric
( gdt . tobit ) allops
All operators behave normally in transformations
( gdt # tobit ) add y2 = y < 20
Expression returns wrong size value
Returns 1x1
Should be 100x1
```

In this example, *y2* will be a scalar because **allops** is in effect. An error message is issued because *y2* does not have the correct number of rows (observations).

See also

dotops

Purpose

Combines all variables in a data set into a matrix.

Format

cat *matrix_name* [[*name_vector*]]

Output

$N \times K$ matrix with columns corresponding to variables.

Remarks

Creates a matrix in the data set workspace containing all the variables in the data set in the same order as the **lv** command variable list.

If *name_vector* is specified, a $K \times 1$ string array is created containing the names of the variables.

Use this command to combine the individual vectors into a single matrix to facilitate operations that can be done more easily with matrix operations.

Use **split** to move the data from the matrix back into the individual vectors.

Code Example

```
cat max0 vname
```

Session Example

This example standardizes the variables in a data set.

```
( gdt maxfact ) cat max0
( gdt maxfact ) g max0 = (max0 - meanc(max0)') ./ stdc(max0)'
Using maxfact workspace
( gdt maxfact ) stats
>   maxfact          4 vars,     100 obs    /data/maxfact.dat

 Variable    Mean  Std Dev  Variance  Minimum  Maximum  Valid Missing
-------------------------------------------------------------------------
  Y1       0.0275   1.0706    1.1461  -3.1796   2.8096    100       0
  Y2       0.0041   0.8206    0.6734  -1.4813   2.7042    100       0
  Y3       0.1155   0.9598    0.9212  -1.9176   3.8984    100       0
  Y4       0.1058   0.8382    0.7025  -2.1307   2.1470    100       0
```

```
( gdt maxfact ) split max0
( gdt * maxfact ) stats
> * maxfact         4 vars,     100 obs    /data/maxfact.dat

 Variable    Mean  Std Dev  Variance  Minimum  Maximum  Valid Missing
-----------------------------------------------------------------------
* Y1        0.0000   1.0000    1.0000  -2.9958   2.5987    100       0
* Y2       -0.0000   1.0000    1.0000  -1.8101   3.2904    100       0
* Y3       -0.0000   1.0000    1.0000  -2.1182   3.9413    100       0
* Y4       -0.0000   1.0000    1.0000  -2.6682   2.4353    100       0
( gdt * maxfact )
```

The data set does not change until the **split** command is executed, as can be seen by the output of the two **stat** commands in the above example.

See also

split

Purpose

Changes the working directory.

Format

cd *file_path*

Remarks

The directory is changed to the directory specified.

Code Example

```
cd /file/path
```

Purpose

Removes from a workspace all symbols that are not data set variables.

Format

cleanup

Remarks

When **GDT** opens a data set, the data are loaded into a workspace as individual vectors. As certain commands are executed, extraneous symbols, such as data, procedures, and new variables, are left in the workspace. These extraneous symbols can be removed with the **cleanup** command, which deletes all symbols except variables that are part of the opened data set, or created with the **add** or **code** statements.

Code Example

```
cleanup
```

Purpose

Closes an open data set.

Format

close ⟦*handle, handle, ...*⟧

Remarks

If a *handle* is not specified, the active data set is closed.

If there are uncommitted changes the user will be prompted to save or discard the changes or cancel the close.

Code Example

```
close
```

Session Example

```
( gdt freqdata ) close
( gdt )
```

See also

open, **use**, **nouse**

Purpose

Creates a new variable with discrete values.

Format

code ⟦*vtype*⟧ *newvarname* **with**

value1 **for** *expression1*,

value2 **for** *expression2*,

⋮

valueN **for** *expressionN*

⟦ **default** *default_value_expression* ⟧

Valid vtypes are:

#	numeric (default)
$	character
%	date

Remarks

value in the **for** clause and *default* must be a scalar value or a scalar-returning expression.

The *expression* in the **for** clause must return a true or false (non-zero or 0) result. If the expression returns true (non-zero), the new variable is assigned the *value* in that **for** clause, otherwise, the new variable will be assigned the *default* value if specified or a value of 0.

The **code** statement syntax requires multiple-line entries. The procedure for entering and working in multi-line mode is described in *User Interfaces* (Section 1.3).

Commas are required between each **for** expression; no comma is allowed after the last **for** expression.

Code Example

```
code # agecat with
   1 for age < 21,
   2 for age >= 21 and age < 30,
   3 for age >= 31 and age < 40,
   4 for age >= 41 and age < 50
   default 5
```

■ Session Example

```
( gdt garch ) report freq

CASES PROCESSED BY THIS PROCEDURE:

  100 cases were kept out of 100.
  0 deleted because of missing values.

Y                            2.42655 |*
                                     |****
Valid       100.0000                 |******
Missing       0.0000                 |******************
Mean          0.9821                 |*****************
Std Dev       0.6008                 |*******
Variance      0.3610                 |****
Mode         -1.0168                 |*
Minimum      -1.0168                 |*
Maximum       2.4265        -1.01679 |**

------------------------------------------------------------
  Total N   100
------------------------------------------------------------

( gdt garch ) \
> code newy with
> 1 for y < -.5,
> 2 for y >= -.5 and y < .5,
> 3 for y >= .5
> /

( gdt * garch ) \
> report freq
> use newy
> /

CASES PROCESSED BY THIS PROCEDURE:
```

```
  100 cases were kept out of 100.
  0 deleted because of missing values.

newy                        3.00000 |*****************
                                    |
Valid       100.0000                |
Missing       0.0000                |
Mean          2.8200                |***
Std Dev       0.4579                |
Variance      0.2097                |
Mode          3.0000                |
Minimum       1.0000                |
Maximum       3.0000        1.00000 |*

                  Percents                    Percents
  Value  Count   Cell     Cum   Value  Count   Cell    Cum
-------------------------------------------------------------
      1      3   3.00    3.00       3     85  85.00 100.00
      2     12  12.00   15.00
-------------------------------------------------------------
Total N  100
-------------------------------------------------------------

( gdt * garch )
```

See also

recode

Purpose

Saves current or specified data set changes to the disk.

Format

commit [[*handle*]]

Remarks

If *handle* is not specified, the active data set is committed.

Executing a **commit** makes modifications to the data set permanent.

Code Example

```
commit simdata
```

Session Example

```
( gdt ) open maxfact
( gdt maxfact ) add y5 = y1 + y2
( gdt * maxfact ) commit
5 variables and 100 observations written to /data/maxfact.dat
( gdt maxfact )
```

See also

rollback

Purpose

Copies variables to or from the active data set.

Format

copy ⟦*var1, var2, ...*⟧ **from** *handle*

copy ⟦*var1, var2, ...*⟧ **to** *handle*

Remarks

If no variables are specified, all variables in the source are copied.

Code Example

```
copy age from freqdata

copy X1 to simdata
```

Session Example

```
( gdt . ) open examples/maxsimeq
( gdt . maxsimeq ) open examples/tobit
( gdt . tobit ) copy y2 from maxsimeq
1 variables copied
( gdt . * tobit ) lv
> * tobit                    5 vars,     100 obs,  /gauss/examples/tobit.dat
  Y                             numeric
  CNST                          numeric
  X1                            numeric
  X2                            numeric
* Y2                            numeric
( gdt . * tobit ) copy cnst to maxsimeq
1 variables copied
( gdt . * tobit ) use maxsimeq
( gdt . * maxsimeq ) lv
> * maxsimeq                  5 vars,     100 obs,  /gauss/examples/maxsimeq.dat
  Y1                            numeric
  Y2                            numeric
  X1                            numeric
  X2                            numeric
* CNST                          numeric
( gdt . * maxsimeq )
```

Purpose

Creates an empty data set.

Format

create *data_set_name*

Output

A data set with no variables is added to the **GAUSS** session.

Remarks

If no arguments are included, all open data sets are listed.

Use the **add** or **code** statements to create variables in the data set.

The data set is written to disk only when the commit statement is executed; the data set is written as a .dat file.

The first variable added to the new data set establishes the number of observations and subsequently added variables must have the same number of observations.

Code Example

```
create newdat
```

Session Example

```
( gdt . ) create newdat
( gdt . ) add y = rndn(100,1)
( gdt . * newdat ) lv
> * newdat                        1 vars,     100 obs,  newdat.dat
* y                                 numeric

( gdt . * newdat ) report freq

 CASES PROCESSED BY THIS PROCEDURE:

   100 cases were kept out of 100.
   0 deleted because of missing values.
```

```
y                              3.06419 |**
                                       |*
Valid          100.0000                |****
Missing          0.0000                |********
Mean            -0.0683                |*************
Std Dev          1.1759                |******************
Variance         1.3826                |********
Mode            -3.2408                |**
Minimum         -3.2408                |***
Maximum          3.0642          -3.24079 |*


-----------------------------------------------------------------------
  Total N   100
-----------------------------------------------------------------------

( gdt . * newdat ) commit
( gdt . newdat )
```

See also

add

Purpose

Deletes observations based on an expression.

Format

delete where *expression*

Code Example

```
delete where age < 21
```

Session Example

```
( gdt maxfact ) delete where y5 < -3.9
Deleted 1 rows from maxfact, 99 are left
( gdt * maxfact )
```

Purpose

Turns on interpretation of scalar-returning operators as element by element operators.

Format

dotops

Remarks

Non-dot (scalar-returning) operators in a transformation expression will be interpreted as dot operators (element by element).

Code inside of procedures is not affected by this setting.

For more information, refer to *Scalar and Element by Element Operations* (Section 1.4) and *Element by Element Operators* in the **GAUSS Users Guide**.

Code Example

```
dotops
```

Session Example

```
( gdt # tobit ) dotops
Operators are dot operators in transformations
( gdt . tobit ) add y2 = y < 20
( gdt . * tobit ) add y3 = y .< 20
( gdt . * tobit )
```

In the example above, *y2* and *y3* will be identical.

See also

allops

Purpose

Drops variables from a data set.

Format

drop *var1*[[*, var2, ...*]]

Code Example

```
drop x1,x2
```

Session Example

```
( gdt maxfact ) drop y5, y6
2 variables dropped
( gdt * maxfact )
```

See also

keep

Purpose

Translates a **GAUSS** data set to a data file of a different format.

Format

export *export_type*

Remarks

The export statement syntax requires multiple line entries. Procedures for entering multiple line entries are described in *User Interfaces* (Section 1.3).

For available export types enter:

```
help export
```

For information on a particular export type enter:

```
help export export_type
```

For more discussion see Section 3.1.

Code Example

```
export ascii
```

Session Example

```
( gdt freqdata ) \
1 export ascii
2 output fdata.asc
3 keep age,pay,sex
4 writevarnames
5 /
```

Purpose

Changes the file name associated with the active data set.

Format

frename *new_data_set_name*

Remarks

The active data set is saved to disk under *new_data_set_name*. The original data set file is left unchanged as of the last **commit**.

To change the name of a data set file, use the operating system command for renaming files.

Code Example

```
frename lgtsim
```

Session Example

```
( gdt maxfact ) frename /data/factor
4 variables and 100 observations written to /data/factor.dat
( gdt factor )
```

See **vrename**, **remove**, **recover**

Purpose

Executes **GAUSS** commands and statements in the active workspace.

Format

g *gauss_command_or_statement*

Remarks

Separate multiple statements with semi-colons (;).

allops and **dotops** do not effect code executed with the **g** statement.

Session Example

```
( gdt . ) open cmlfact
( gdt . cmlfact ) cat fdata
( gdt . cmlfact ) g r = corrx(fdata)
Using cmlfact workspace
( gdt . cmlfact ) g print r
Using cmlfact workspace

       1.0000000       0.66830903       0.32650504       0.21293355
      0.66830903        1.0000000       0.30232712       0.24065234
      0.32650504       0.30232712        1.0000000       0.59052208
      0.21293355       0.24065234       0.59052208        1.0000000

( gdt . cmlfact ) g { p,v,a } = princomp(fdata,4)
Using cmlfact workspace
( gdt . cmlfact ) g print v
Using cmlfact workspace

      0.55622722
      0.27164763
     0.094103380
     0.078021774
( gdt . cmlfact )
```

See also

cat, **split**

Purpose

Translates a data file to a **GAUSS** data set.

Format

import *import_type*

Remarks

The import statement syntax requires multiple line entries. Procedures for entering multiple line entries are described in *User Interfaces* (Section 1.3).

For available import types enter:

```
help import
```

For information on a particular import type enter:

```
help import import_type
```

For more discussion see Section 2.1.

Code Example

```
import ascii
```

Session Example

```
( gdt ) \
1 import ascii
2 input mydata.asc
3 output mydata
4 invarline 1
5 vartypeline 2
6 skip 2
7 /
```

Purpose

Imputes missing data.

Format

impute *impute_method*

Remarks

The impute statement syntax requires multiple line entries. The procedure for entering multiple line entries is described in *User Interfaces* (Section 1.3).

For available imputation methods enter:

```
help impute
```

For information on a particular method enter:

```
help impute impute_method
```

Code Example

```
impute ems
```

Session Example

```
( gdt . ) open freqdata
( gdt . freqdata ) stats

================================================================================================================
>   freqdata            4 vars,     400 obs    /gauss/examples/freqdata.dat

 Variable                  Mean       Std Dev        Variance       Minimum       Maximum       Valid     Missing
----------------------------------------------------------------------------------------------------------------
  AGE                    5.6784        2.9932          8.9593        1.0000       10.0000         398           2
  PAY                    1.9675        0.8019          0.6431        1.0000        3.0000         400           0
  sex                     -----         -----           -----         -----         -----       -----       -----
  WT                     1.4699        0.3007          0.0904        1.0000        1.9900         400           0

( gdt . freqdata ) impute ems

WARNING: variable No. 3 may be categorical.  Analysis
will continue with this variable coded to a sequence,
However, it may not conform to the assumptions of the
missing data model which requires Normality.
( gdt . * freqdata ) stats

================================================================================================================
> * freqdata            4 vars,     400 obs    /gauss/examples/freqdata.dat

 Variable                  Mean       Std Dev        Variance       Minimum       Maximum       Valid     Missing
----------------------------------------------------------------------------------------------------------------
```

```
* AGE                  5.6785        2.9858        8.9147        1.0000       10.0000          400            0
* PAY                  1.9675        0.8019        0.6431        1.0000        3.0000          400            0
* sex                  1.3850        0.4872        0.2374        1.0000        2.0000          400            0
* WT                   1.4699        0.3007        0.0904        1.0000        1.9900          400            0
( gdt . * freqdata )
```

Purpose

Drops all variables except those listed.

Format

keep *variable1*[[*, variable2, ...*]]

Code Example

```
keep y1, y2
```

Session Example

```
( gdt factor )  lv
>   factor    4 vars,     100 obs,  /data/factor.dat
  Y1             numeric
  Y2             numeric
  Y3             numeric
  Y4             numeric
( gdt factor ) keep y1,y2
2 variables kept
( gdt * factor )  lv
>   factor    2 vars,     100 obs,  /data/factor.dat
  Y1             numeric
  Y2             numeric
```

See also

drop

Purpose

Lists data sets that are on disk.

Format

ld *file_path*

Remarks

Files in the current directory are displayed if *file_path* is not specified.

Code Example

```
ld /data/examples
```

Session Example

```
( gdt factor )   ld
garch       free,    1 vars,  100 obs,  /data/garch.dat
linear      free,    5 vars,  100 obs,  /data/linear.dat
logit2      free,    3 vars, 1000 obs,  /data/logit2.dat
maxfact     free,    4 vars,  100 obs,  /data/maxfact.dat
sci         free,   23 vars,  154 obs,  /data/sci.dat
( gdt factor )
```

Purpose

Lists variables in the active or specified data set.

Format

lv [[*handle*]]

Remarks

The variables are listed in the order they appear in the data set.

Code Example

```
lv

lv histdata
```

Session Example

```
( gdt freqdata ) lv
>   freqdata     4 vars,     400 obs,  /data/freqdata.dat
  AGE               numeric
  PAY               numeric
  sex               character
  WT                numeric
( gdt freqdata )
```

Purpose

Merges data sets on a key variable or list of key variables.

Format

merge from *handle* **on** *keyvar1*, *keyvar2*, ... ⟦**variables** *var1*, *var2*, ...⟧

Remarks

The active data set is modified by the merge while the other is left unchanged.

Data sets to be merged must have at least one common key variable between them. Only observations that match on key variables in both data sets are included in the merge. All other observations are deleted from the active data set.

Key variables in both data sets must be sorted unique before merging.

If no **variables** are specified, all variables from the *handle* not in the key variable list are merged into the active data set, otherwise, only the specified variables from the *handle* are merged.

Merged variables with duplicate names in both the *handle* and active data set are renamed.

Code Example

```
merge from empdata on empid variables department, hiredate, position
```

Session Example

```
( gdt . ) open sim1
( gdt . sim1 ) lv
>  sim1                          2 vars,     100 obs,  /data/sim1.dat
  ky                                numeric
  y1                                numeric

( gdt . sim1 ) open sim2
( gdt . sim2 ) lv
>  sim2                          2 vars,     100 obs,  /data/sim2.dat
  ky                                numeric
  y2                                numeric

( gdt . sim2 ) merge from sim1 on ky
( gdt . * sim2 ) lv
```

```
> * sim2                          3 vars,     100 obs,  /data/sim2.dat
* ky                                numeric
* y2                                numeric
* y1                                numeric
( gdt . * sim2 )
```

See also

copy, **sort**, **stack**

Purpose

Creates a data set from a model simulation.

Format

model *model_name*

Output

A new data set containing simulated data.

Remarks

The model statement syntax requires multiple line entries. Procedures for entering multiple line entries are described in *User Interfaces* (Section 1.3).

For available models enter:

```
help model
```

For information on a particular model enter:

```
help model model_name
```

Code Example

```
model probit
```

Session Example

```
( gdt ) \
1 model arima
2 file test
3 depvar Y
4 indvar X1,X2
5 vcx 1,.4,1
6 ar .3,.1
7 ma .2
8 normal 1.5
9 open
10 /

( gdt . test ) report freq
```

```
CASES PROCESSED BY THIS PROCEDURE:

  100 cases were kept out of 100.
  0 deleted because of missing values.


Y                             3.92054 |*
                                      |***
Valid          100.0000               |*******
Missing          0.0000               |******
Mean             0.9487               |*****************
Std Dev          1.0865               |******************
Variance         1.1804               |*********
Mode            -1.6957               |********
Minimum         -1.6957               |**
Maximum          3.9205        -1.69574 |*****

------------------------------------------------------------------
  Total N   100
------------------------------------------------------------------


X1                            2.77424 |**
                                      |*
Valid          100.0000               |******
Missing          0.0000               |*******
Mean             0.1678               |***************
Std Dev          0.9554               |******************
Variance         0.9128               |**********
Mode            -1.4980               |************
Minimum         -1.4980               |**************
Maximum          2.7742        -1.49800 |*********

------------------------------------------------------------------
  Total N   100
------------------------------------------------------------------


X2                            2.88571 |**
                                      |*
Valid          100.0000               |*****
Missing          0.0000               |*******
```

```
Mean           0.0246                 |****************
Std Dev        0.9947                 |******************
Variance       0.9894                 |***************
Mode          -2.1527                 |*********
Minimum       -2.1527                 |****
Maximum        2.8857         -2.15270 |******

-----------------------------------------------------------------
  Total N   100
-----------------------------------------------------------------

( gdt . test )
```

Purpose

Makes the active data set not active.

Format

nouse

Remarks

It is possible for no open data sets to be active.

Code Example

```
nouse freqdata
```

Session Example

```
( gdt freqdata ) nouse freqdata
( gdt )
```

See also

use, **open**, **close**

Purpose

Loads a data set into a workspace and makes it the active data set.

Format

open *data_set_name*

Remarks

Each open data set has its own workspace. The data are loaded into the workspace as K $Nx1$ vectors. Vector names are the same as the data set variable names.

If no arguments are included, a list of open data sets is displayed on the screen.

Code Example

```
open examples/freqdata
```

See also

close, **use**, **nouse**

Purpose

Prints samples of one or more variables to the screen.

Format

p ⟦*numobs*⟧ ⟦*var*⟧

Remarks

If no arguments are specified, the first *numobs* observations of each variable in the data set are displayed on the screen. The default value of *numobs* is 50. To change the default value, enter the *numobs* parameter with no variable name. The default will remain changed until **GDT** is restarted.

If a variable name is specified, the first *numobs* observations of the specified variable are displayed, and options are given for viewing other observations within that variable.

Code Example

```
p 100 age
```

Session Example

```
( gdt tobit ) p 10
( gdt tobit ) p

Y =

[1:5]       0.19122822  -0.52539158  -0.15540838            0   -1.2125193
[6:10]               0   0.31077457    1.1137943  -0.93615974   0.80979624

CNST =

[1:10]     1  1  1  1  1  1  1  1  1  1

X1 =

[1:5]      0.59209035  0.25336661  0.88820146 -0.85628518  0.25278613
[6:10]     0.32553164 -0.036312038  0.94893974  -1.3409896 -0.22219151

X2 =

[1:5]      -0.35181439  0.025950747    1.7658545    1.0476031  -0.78446166
[6:10]     -0.12405032 -0.022460097   0.37790726  -0.79498798  -0.24560567
```

```
( gdt tobit ) p 25 y

Y =

[1:5]       0.19122822  -0.52539158  -0.15540838            0   -1.2125193
[6:10]               0   0.31077457    1.1137943  -0.93615974   0.80979624
[11:15]    -0.79397375   0.56180099  -0.20987604   0.71051733  -0.68155363
[16:20]     0.98908776    1.0933305   0.04909111  -0.92224655    1.4741197
[21:25]     -1.0520466    1.3088757      2.41852   0.13847887    2.1946464
( Next | Previous | Top | Bottom | ### | Show ### | Help | Quit ) [ N ] n

Y =

[26:30]      0.22283408  -0.35802955  0.078518682  -0.30831094  -0.28662837
[31:35]     -0.63163611            0  -0.16819537    1.0455784   0.28637601
[36:40]     -0.19024424   0.19664649  -0.97867385   0.27319029   0.39976644
[41:45]      0.60695266   0.10849636  0.038987486     1.044837            0
[46:50]      0.52982032  -0.92003283  -0.11145973  -0.69712074   0.63632036
( Next | Previous | Top | Bottom | ### | Show ### | Help | Quit ) [ N ] q
( gdt tobit )
```

Purpose

Shows the current directory.

Format

pwd

Code Example

```
pwd
```

Session Example

```
( gdt . tobit ) pwd
Current directory: /data/examples
( gdt . tobit )
```

Purpose

Exits **GDT**

Format

q

quit

Code Example

```
q

quit
```

Session Example

```
( gdt . tobit ) q
(/data/example)%

( gdt . tobit ) quit
(/data/example)%
```

Purpose

Recodes a variable to discrete values.

Format

recode [[*vtype*]] *varname* **with**

value1 **for** *expression*,

value2 **for** *expression*,

⋮

valueN **for** *expression*

Valid vtypes are:

#	numeric
$	character
%	date

Remarks

If *vtype* is not specified, the type of the variable is unchanged.

value in the **for** clause must be a scalar value or a scalar-returning expression.

The *expression* in the **for** clause must return a true or false (non-zero or 0) result; if the expression returns true (non-zero), the variable is assigned the *value* in that **for** clause, otherwise, the variable value remains unchanged.

The **recode** statement syntax requires multiple-line entries. The procedure for entering and working in multi-line mode is described in *User Interfaces* (Section 1.3).

Commas are required between each **for** expression; no comma is required after the last **for** expression.

Code Example

```
recode # age with
   1 for age < 21,
   2 for age >= 21 and age < 30,
```

```
   3 for age >= 31 and age < 40,
   4 for age >= 41 and age < 50,
   5 for age >= 50
```

Session Example

```
( gdt . gssocc ) \
   1 recode educ with
   2 1 for educ <= 8,
   3 2 for educ > 8 and educ <= 12,
   4 3 for educ > 12
   5 /
( gdt . * gssocc ) \
   1 report freq
   2 use educ
   3 /

CASES PROCESSED BY THIS PROCEDURE:

  337 cases were kept out of 337.
   0 deleted because of missing values.



educ                          3.00000 |******************
                                      |
Valid          337.0000               |
Missing          0.0000               |
Mean             2.4154               |****************
Std Dev          0.6264               |
Variance         0.3924               |
Mode             3.0000               |
Minimum          1.0000               |
Maximum          3.0000          1.00000 |***

                    Percents                          Percents
    Value   Count     Cell      Cum     Value   Count     Cell      Cum
-------------------------------------------------------------------------
        1      25     7.42     7.42         3     165    48.96   100.00
        2     147    43.62    51.04
-------------------------------------------------------------------------
  Total N   337
-------------------------------------------------------------------------

( gdt . * gssocc )
```

See also

code

Purpose

Recovers a data set that has been removed.

Format

recover *data_set_name*

Remarks

The file associated with the removed data set is renamed by dropping the .removed extension; the .removed file must still be on the disk.

See also

remove

Purpose

Closes and renames the specified data set.

Format

remove *handle*

Remarks

The data set must be open.

The data set is closed and the associated file is renamed by appending a .removed extension; any previous extenson in the filename remains intact.

To delete a data set file from the disk, use the operating system commands for deleting files.

Code Example

```
remove test
```

Session Example

```
( gdt test ) remove test
/data/test.dat renamed to /data/test.dat.removed
( gdt )
```

See also

recover

Purpose

Presents data in various formats such as listings, tables, and plots.

Format

report *report_name*

Remarks

The report statement syntax requires multiple line entries. Procedures for entering multiple line entries are described in *User Interfaces* (Section 1.3).

For available reports enter:

```
help report
```

For information on a particular report enter:

```
help report report_name
```

Code Example

```
report freq
```

Session Example

```
( gdt freqdata ) \
1 report freq
2 use age,pay,sex
3 weight wt
4 /


CASES PROCESSED BY THIS PROCEDURE:

  398 cases were kept out of 400.
  2 deleted because of missing values.



age                          10.00000 |******************
                                      |****************
Valid       585.5600                  |***************
```

```
Missing       0.0000                |************
Mean          5.6530                |****************
Std Dev       2.9763                |***************
Variance      8.8584                |*********
Mode         10.0000                |*****************
Minimum       1.0000                |***********
Maximum      10.0000         1.00000 |*****************
```

Value	Count	Percents Cell	Percents Cum	Value	Count	Percents Cell	Percents Cum
1	67.94	11.60	11.60	6	62.25	10.63	58.04
2	44.42	7.59	19.19	7	49.54	8.46	66.50
3	67.52	11.53	30.72	8	58.63	10.01	76.51
4	36.66	6.26	36.98	9	65.77	11.23	87.75
5	61.07	10.43	47.41	10	71.76	12.25	100.00

Total N 586

```
pay                          3.00000 |***************
                                     |
Valid         587.9800               |
Missing         0.0000               |
Mean            1.9748               |******************
Std Dev         0.8008               |
Variance        0.6413               |
Mode            2.0000               |
Minimum         1.0000               |
Maximum         3.0000         1.00000 |*****************
```

Value	Count	Percents Cell	Percents Cum	Value	Count	Percents Cell	Percents Cum
1	195.8	33.30	33.30	3	180.98	30.78	100.00
2	211.2	35.92	69.22				

Total N 588

```
sex                              F |******************
                                 M |************
Valid         587.9800
Missing         0.0000
```

```
Mode               F

                   Percents                          Percents
 Value   Count    Cell     Cum    Value   Count    Cell     Cum
-------------------------------------------------------------------
     F  360.92   61.38   61.38        M  227.06   38.62  100.00
-------------------------------------------------------------------
  Total N   588
-------------------------------------------------------------------

( gdt freqdata )
```

Purpose

Discards all changes since the last commit.

Format

rollback ⟦*handle*, *handle*, ...⟧

Remarks

All changes to the data set since the last **commit** are discarded.

If *handle* is not specified, the active data set is rolled back.

The **frename** command is not rolled back.

Code Example

```
rollback
```

```
rollback hist, proj
```

See also

commit

Purpose

Creates a random sample of observations in a data set.

Format

sample *num*% [[**without**]]

sample *num_obs* [[**without**]]

Remarks

This command modifies the data set by deleting observations; consider using **frename** before executing **sample**.

If % is appended to *num*, the sample will consist of the specified percentage of observations, otherwise, the data set is modified to contain *num_obs* observations.

If **without** is added, sampling is without replacement, otherwise, it is with replacement.

Code Example

sample 10%

sample 1000

Session Example

```
( gdt freqdata ) stats
>   freqdata     4 vars,     400 obs    /data/freqdata.dat

 Variable       Mean   Std Dev  Variance  Minimum  Maximum  Valid Missing
-------------------------------------------------------------------------
  AGE         5.6784    2.9932    8.9593   1.0000  10.0000    398       2
  PAY         1.9675    0.8019    0.6431   1.0000   3.0000    400       0
  sex          -----     -----     -----    -----    -----  -----   -----
  WT          1.4699    0.3007    0.0904   1.0000   1.9900    400       0
( gdt freqdata ) sample 10%
( gdt * freqdata ) stats
> * freqdata       4 vars,      40 obs    /data/freqdata.dat

 Variable       Mean   Std Dev  Variance  Minimum  Maximum  Valid Missing
-------------------------------------------------------------------------
* AGE         6.4872    2.8365    8.0459   1.0000  10.0000     39       1
* PAY         1.9750    0.8002    0.6404   1.0000   3.0000     40       0
```

```
* sex          -----     -----     -----     -----     -----  -----   -----
* WT          1.4400    0.3133    0.0981    1.0000    1.9800     40       0
( gdt * freqdata )
```

See also

frename

Purpose

Selects observations based on an expression.

Format

select where *expression*

Remarks

This command deletes all observations that do not match the expression.

Code Example

```
select where age >= 21 and age <= 45
```

Session Example

```
( gdt . * tobit ) select where y <= -3.9
Selected 61 of 61 rows from tobit, 0 rows dropped
( gdt . * tobit )
```

See also

delete

Purpose

Shows the content of the active data set workspace.

Format

show

Remarks

Each open data set has its own workspace.

Use **cleanup** to remove all data in the workspace that are not variables of the data set.

Code Example

```
show
```

Session Example

```
( gdt . nlls ) show

     1200 bytes     T          MATRIX                       150,1
     1200 bytes     Y          MATRIX                       150,1
```

See also

cleanup

Purpose

Sorts the active data set.

Format

sort ⟦*unique*⟧ **on** *var1, var2, ...*

Remarks

Use *unique* to exclude duplicate observations—where more than one observation contains the same key variable value, only one observation will be included.

Code Example

```
sort on age, date

sort unique on age, date
```

Purpose

Splits a matrix into individual vectors.

Format

split *name_of_matrix*

Remarks

Splits a matrix in the data set workspace into individual vectors. The matrix must contain as many columns as there are variables in the data set and the columns must be in the same order as the **lv** variable list.

The number of observations in the data set may change but the number of variables will not.

Code Example

```
split statdat
```

Session Example

```
( gdt maxfact ) cat max0
( gdt maxfact ) g max0 = (max0 - meanc(max0)') ./ stdc(max0)'
Using maxfact workspace
( gdt maxfact ) stats
>   maxfact          4 vars,     100 obs    /data/maxfact.dat

Variable    Mean  Std Dev  Variance  Minimum  Maximum  Valid Missing
-------------------------------------------------------------------
Y1       0.0275   1.0706    1.1461  -3.1796   2.8096    100       0
Y2       0.0041   0.8206    0.6734  -1.4813   2.7042    100       0
Y3       0.1155   0.9598    0.9212  -1.9176   3.8984    100       0
Y4       0.1058   0.8382    0.7025  -2.1307   2.1470    100       0

( gdt maxfact ) split max0
( gdt * maxfact ) stats
>  * maxfact         4 vars,     100 obs    /data/maxfact.dat

Variable    Mean  Std Dev  Variance  Minimum  Maximum  Valid Missing
-------------------------------------------------------------------
* Y1       0.0000   1.0000    1.0000  -2.9958   2.5987    100       0
* Y2      -0.0000   1.0000    1.0000  -1.8101   3.2904    100       0
* Y3      -0.0000   1.0000    1.0000  -2.1182   3.9413    100       0
```

```
* Y4       -0.0000   1.0000    1.0000  -2.6682   2.4353    100        0

( gdt * maxfact )
```

See also

cat

Purpose

Appends observations from the specified data set to the active data set.

Format

stack *handle*

Remarks

Only variables in the source data set with the same names as variables in the active data set will be copied.

This command will not work if the active data set contains variables that are not also contained in the source data set.

Code Example

```
stack gssocc
```

Session Example

```
( gdt . ) open maxsimeq
( gdt . maxsimeq ) open maxnleq
( gdt . maxnleq ) stats

================================================================================
>   maxnleq       4 vars,     100 obs    maxnleq.dat

 Variable   Mean  Std Dev Variance  Minimum  Maximum  Valid  Missing
--------------------------------------------------------------------------------
  Y1      1.4516   1.9119   3.6553  -2.4473   8.3456    100       0
  Y2      1.4347   1.9034   3.6227  -1.7680   7.7942    100       0
  X1      0.0581   0.9999   0.9997  -2.1374   2.2291    100       0
  X2     -0.0789   1.0497   1.1019  -2.3797   2.0961    100       0
( gdt . maxnleq ) stack maxsimeq
( gdt . * maxnleq ) stats

> * maxnleq       4 vars,     200 obs   maxnleq.dat

 Variable    Mean  Std Dev Variance  Minimum  Maximum Valid  Missing
--------------------------------------------------------------------------------
* Y1       1.3815   1.5549   2.4177  -2.4473   8.3456   200       0
* Y2       1.4360   1.5223   2.3174  -1.7680   7.7942   200       0
* X1       0.0324   1.0110   1.0222  -2.6835   2.9954   200       0
```

```
* X2      -0.0723   1.0445   1.0909  -3.0756   2.5930   200       0
( gdt . * maxnleq )
```

Purpose

Computes statistics on the active data set.

Format

stats ⟦*var1, var2, ...*⟧

Remarks

If no variables are given, all variables are included.

If no data set is active, statistics are generated for all open data sets.

Code Example

```
stats

stats age, pay
```

Session Example

```
( gdt . maxnleq ) stats

>   maxnleq       4 vars,     100 obs    maxnleq.dat

 Variable   Mean  Std Dev Variance  Minimum  Maximum  Valid  Missing
-----------------------------------------------------------------------
  Y1      1.4516   1.9119   3.6553  -2.4473   8.3456    100        0
  Y2      1.4347   1.9034   3.6227  -1.7680   7.7942    100        0
  X1      0.0581   0.9999   0.9997  -2.1374   2.2291    100        0
  X2     -0.0789   1.0497   1.1019  -2.3797   2.0961    100        0
( gdt . maxnleq )
```

Purpose

Makes the specified data set the active data set.

Format

use *handle*

Remarks

To see open data sets, use the open statement with no arguments.

Code Example

```
use freqdata
```

Session Example

```
( gdt ) use freqdata
( gdt freqdata )
```

See also

nouse, **open**, **close**

Purpose

Renames a variable.

Format

vrename *old_name new_name*

Code Example

```
vrename hst newhst
```

Session Example

```
( gdt . ) open hensher
( gdt . hensher ) lv
>   hensher                      8 vars,     840 obs,  hensher.dat
  MODE                             numeric
  TTME                             numeric
  INVC                             numeric
  INVT                             numeric
  GC                               numeric
  HINC                             numeric
  PSIZE                            numeric
  AIRHINC                          numeric
( gdt . hensher ) vrename ttme traveltime
( gdt . * hensher ) lv
> * hensher                      8 vars,     840 obs,  hensher.dat
  MODE                             numeric
* traveltime                       numeric
  INVC                             numeric
  INVT                             numeric
  GC                               numeric
  HINC                             numeric
  PSIZE                            numeric
  AIRHINC                          numeric
( gdt . * hensher )
```

See also

open

Purpose

Changes the variable type of a variable.

Format

vtype *new_type var1*[[*, var2, ...*]]

Valid types are:

#	numeric
$	character
%	date

Code Example

```
vtype # sex
```

Session Example

```
( gdt . freqdata ) sex = sex $== "F"
( gdt . * freqdata ) vtype # sex
1 variables set to type numeric
( gdt . * freqdata ) lv
> * freqdata                    4 vars,     400 obs,  freqdata.dat
  AGE                             numeric
  PAY                             numeric
* sex                             numeric
  WT                              numeric
( gdt . * freqdata )
```

See also

open

Index

I

K

L

M

N

O

P

Q

R

S

T

U

V

W