The Elements of STATA

Numbers

A number may contain a sign, an integer part, a decimal point, a fraction part, an **e** or **E**, and a signed integer exponent. Numbers may not contain commas; for example, the number 1,024 must be typed as 1024 (or 1024. or 1024.0 or ...). The following are examples of valid numbers:

```
5
-5
5.2
.5
5.2e+2
5.2e-2
```

A number can also take on the special value "missing", denoted by a single period (.). You may specify a missing value any place you may specify a number. Do not place the period in double quotes or STATA will interpret it as a string. Missing values differ from ordinary numbers in one respect: any arithmetic operation on a missing value yields a missing value.

Technical Note: Numbers can be stored in one of four variable types: **int**, **long**, **float** (the default), or **double**. **int**s are stored in 2 bytes, **long**s and **float**s in 4 bytes, and **double**s in 8 bytes. **int**s may contain any number between -32,768 and 32,766 inclusive, and missing values are stored as 32,767. **long**s may contain any number between -2,146,483,648 and 2,147,483,646 inclusive, and missing values are stored as 2,147,483,647. **float**s may contain any number between +/-10^-37 and +/-10^37, and missing values are stored as 2^128. **double**s may contain any number between +/-10^-99

and +/-10^99, and missing values are stored as 2^333.

Do not confuse the term integer, which is a characteristic of a number, with **int**, which is a storage type. For instance, 5 is an integer no matter how it stored. Thus if you read that an argument is required to be an integer that does not mean that it must be stored as an **int**.

Names

A name is a string of one to eight letters (A-Z and a-z), digits (0-9), and underscore (_). STATA reserves the names **double**, **float**, **if**, **in**, **int**, **long**, **using**, **with**, **_all**, **_b**, **_coef**, **_n**, **_N**, **_pi**, and **_rc**. You may not use these reserved names for your variables. The first character of a name must be either a letter or an underscore. We recommend, however, that you do not begin your variable names with an underscore. All STATA built-in variables begin with an underscore, and we reserve the right to incorporate new _variables freely.

STATA respects case, that is, **myvar**, **Myvar**, and **MYVAR** are distinct names in STATA.

Raw Data

Raw data (**data**) is a rectangular table of numeric values where each row is an observation on all the variables and each column contains all the observations on a single variable. Observations are numbered sequentially from 1 to **_N**. The following example of **data** contains the first five odd and the first five even numbers:

Elements

	odd	even
1.	1	2
2.	3	4
3.	5	6
4.	7	8
5.	9	10

The observations are numbered 1 to 5 and the variables are named **odd** and **even**.

Cross-Product

A data set can be stored as a cross-product (**xp**) rather than as **data**. Define X to be a raw data matrix augmented on the left with a system variable (**_cons**) every element of which is equal to one. The **xp** form of this data is the matrix inner product X'X. For the example above, the corresponding **xp** form is:

	_cons	odd	even
1.	5	24	29
2.	24	156	180
3.	29	180	209

Several STATA commands, such as **regress** and **correlate**, execute more rapidly if the data set is stored in **xp** form. In addition, STATA can process a data set with an unlimited number of observations if the data set is stored in **xp** form. The **convert** command can be used to transform **data** format data sets to **xp** data sets. Note that not all STATA commands can be used on **xp** data sets. See the description of the **convert** command for more details.

It is also possible to enter a cross-product directly into STATA (using **input** or **infile**) and then to direct STATA to interpret the data as a cross-product via the **set contents xp** command. If

you use this method, you must be careful that the data you input conforms exactly to the description of **X'X** given above. For example, the value in the first row and first column must contain the number of observations in the data set. When you **input** or **infile** your data, you may give the first variable any name except **_cons**. The **set contents xp** command automatically renames the first variable **_cons** if the data set meets the requirements of a cross-product. **_cons** is the special name STATA reserves for the first column of a cross-product. (Note: As a convenience, STATA allows you to set one, but not both, of a pair of corresponding off-diagonal elements of a cross-product to missing value. The non-missing value will be automatically copied over the missing value by **set contents xp**.)

Technical Note: When STATA creates a cross-product, it stores all the data as **doubles**. If you create your own cross-product directly, we recommend that you do likewise. However, you may use any variable types you wish.

Language Syntax

With few exceptions, the basic language syntax is:

[by varlist:] command [varlist] [=exp]
[if exp] [**in** range] [, options]

where square brackets denote optional qualifiers. In this diagram, "varlist" denotes a list of variable names, "command" denotes a STATA command, "exp" denotes an algebraic expression, "range" denotes an observation range, and "options" denotes a list of options.

Most commands that take a subsequent varlist do not require one to be explicitly typed. If no varlist appears, these commands assume a varlist of **_all**,

the STATA shorthand for indicating all the variables in the data set. In commands that alter or destroy data, STATA always requires that the varlist be specified explicitly.

The **by** varlist: prefix and the **if** exp and **in** range qualifiers are described completely in the Command Reference section of this manual. Briefly, the **by** varlist: prefix causes STATA to repeat a command for each subset of the data for which the values of the variables in the varlist are equal. The **if** exp qualifier restricts the scope of the command to those observations for which the value of the expression is non-zero. The **in** range qualifier restricts the scope of the command to a specific observation range.

The =**exp** phrase serves two different functions. In the **generate** and **replace** commands, =exp specifies the values to be assigned to a variable. In other STATA commands, =exp is used to indicate the weight to attach to each observation. In these latter commands, failing to specify a weight is equivalent to specifying =**1**.

Many commands take command specific options. These are described along with each command in the Command Reference section of this manual.

STATA treats any line starting with a "*" as a comment and ignores it.

Abbreviation Rules

Command, variable, and option names may be abbreviated to the shortest string of characters that uniquely identifies them. For instance, there are four commands that start with the letter "r": **regress**, **rename**, **replace**, and **run**. Therefore **regress** may be abbreviated as **regres**, **regre**, **regr**, or **reg**. It may not be abbreviated as **re** since this string does not distinguish **regress** from **rename** and **replace**.

There is one exception to the abbreviation rule: if a command or option alters or destroys data, then the command or option name must be spelled out completely. For example, the **drop** command may not be abbreviated.

varlists

A varlist is a list of variable names. The variable names in a varlist refer exclusively either to new (not yet created) variables or to existing variables.

In lists of existing variable names, variable names may be repeated in the varlist. The variable names may also be abbreviated. A "*" may be appended to a partial variable name to indicate all variables that start with that letter combination. For example, if the variables pop1t5, pop5to6, and pop18p are in your data set, you may type pop* as a shorthand way to refer to all three variables. You may also place a dash (-) between two variable names to specify all the variables stored between the two listed variables inclusive. (The **describe** command lists variables in the order in which they are stored.)

In lists of new variables, no variable names may be

repeated or abbreviated in the varlist. You may specify a dash (-) between two variable names that have the same letter prefix and that end in numbers. This form of the dash notation indicates a range of variables in ascending numerical order. For instance, typing "v1-v4" is equivalent to typing "v1 v2 v3 v4".

In lists of new variables, you may type the name of a storage type before the variable name to force a storage type other than the default. The storage types are **int**, **long**, **float** (the default), and **double**. For instance, the list "var1 int var2 var3" specifies that var1 and var3 are to be given the default storage type, and var2 is to be stored as an **int**. You may use parentheses to bind a list of variable names. The list "var 1 int(var2 var3)" specifies that both var2 and var3 are to be stored as **ints**.

In lists of new variables, you may also append a colon and a value label name. For instance, "var1 var2:myfmt" specifies that the variable var2 is to be associated with the value labels stored under the name myfmt. This has the same effect as typing the list "var1 var2" and then subsequently giving the command

label values var2 myfmt

The advantage of specifying the value label association with the colon notation is that the value labels can then be used by the current command. (See the descriptions of the **input** and **infile** commands for further explanations of using the colon notation.)

Expressions

STATA includes a complete expression parser. Algebraic expressions are specified in a natural way using the standard rules of hierarchy. For instance, **myvar+2/othvar** is interpreted as **myvar+(2/othvar)**. You may use parentheses freely to force a different order of evaluation.

Operators

The arithmetic operators in STATA are: + (addition), - (subtraction), * (multiplication), / (division), ^ (raise to a power), and the prefix - which indicates negation. Any arithmetic operation on a missing value or any impossible arithmetic operation (such as division by zero) yields a missing value.

The relational operators in STATA are: > (greater than), < (less than), >= (greater than or equal), <= (less than or equal), == (equal), and ~= (not equal). Note that the relational operator for equality is a pair of equal signs. This convention distinguishes relational equality from the =exp phrase.

Relational expressions are either true (denoted by **1**) or false (denoted by **0**). Relational operations are performed after all arithmetic operations. Thus the expression **(3>2)+1** is equal to **2** while **3>2+1** is equal to **0**. Missing values may appear in relational expressions. The expression **x==.** is true (equal to **1**) if x is missing and false (equal to **0**) otherwise. A missing value is greater than any non-missing value.

The logical operators in STATA are: & (and), | (or), and ~ (not). On input, the logical operators interpret any non-zero value (including

missing value) as true and zero as false. Like the relational operators, they return the value **1** for true and **0** for false. For example, the expression **5 & .** is equal to **1**. Logical operations, except for **~**, are performed after all arithmetic and relational operations; the expression **3>2 & 5>4** is interpreted as **(3>2)&(5>4)** and is equal to **1**.

The order of evaluation (from first to last) of all the operators is: **-** (negation), **~**, **^**, **/**, *****, **-** (subtraction), **+**, **~=**, **>**, **<**, **<=**, **>=**, **==**, **&**, **|**.

Functions

Functions may appear in expressions. Functions are indicated by the function name, an open parenthesis, an expression or expressions separated by commas, and a close parenthesis. For example, the square root of a variable named x is specified by typing **sqrt(x)**. All functions return missing values when given missing values as arguments or when the result is undefined.

The mathematical functions in STATA are: **abs**(x) (absolute value), **atan**(x) (arc-tangent returning radians), **cos**(x) (cosine of radians), **exp**(x) (exponent), **mod**(x,y) (the modulus of x with respect to y), **sin**(x) (sine of radians), and **sqrt**(x) (square root).

The statistical functions in STATA are: **chiprob**(df,x) (the cumulative chi-square with df degrees of freedom and value x), **fprob**(df1,df2,f) (the cumulative F-distribution with df1 numerator and df2 denominator degrees of freedom), **invnorm**(p) (the inverse cumulative normal), **normprob**(z) (the cumulative normal), and **tprob**(df,t) (Student's cumulative t-distribution with df degrees of freedom).

STATA includes a random number function, **uniform()**, which takes no arguments (although you must include the open and close parentheses). It produces uniformly distributed pseudo-random numbers over the open interval zero to one. Every time STATA is started, **uniform()** produces the same sequence of numbers. The seed value, and hence the sequence of pseudo-random numbers, can be changed with the **set seed** command.

STATA also includes the following special functions:

autocode(x,ng,xmin,xmax) partitions the interval from xmin to xmax into ng equal length intervals and returns the upper bound of the interval which contains x. This function is an automated version of **recode()** (see below). The algorithm for **autocode** is

```
if (x==. | ng==. | xmin==. | xmax==. | ng<=0
   | xmin>=xmax) then return .
otherwise
   for i=1 to ng-1
     xmap=xmin+i*(xmax-xmin)/ng
     if x<=xmap then return xmap
   end
   otherwise
      return xmax
```

cond(x,a,b) returns a if x evaluates to true (not **0**) and b if x evaluates to false (**0**). For example,

```
generate maxinc=cond(inc1>inc2,inc1,inc2)
```

creates the variable maxinc as the maximum of inc1 and inc2.

float(x) returns the value of x rounded to **float**. Although you may store your variables as **double**,

float, **long**, or **int**, STATA converts all numbers to **double** before performing any calculations. As a consequence, difficulties can arise when comparing numbers that have no finite digit binary representation. For example, if the variable x is stored as a **float** and contains the value 1.1 (a repeating decimal in binary) the expression **x==1.1** will evaluate to false because the literal **1.1** is the **double** representation of 1.1 which is different than the **float** representation stored in x. The expression **x==float(1.1)** will evaluate to true, because the **float** function converts the literal **1.1** to its **float** representation before it is compared to x.

group(x) creates a categorical variable that divides the data into x as near equally sized subsamples as possible, numbering the first group 1, the second 2, and so on.

int(x) returns the integer obtained by truncating x.

max(x1,x2,...,xn) returns the maximum of x1, x2, ..., xn. Missing values are ignored. If all the arguments are missing, missing is returned.

min(x1,x2,...,xn) returns the minimum of x1, x2, ..., xn. Missing values are ignored. If all the arguments are missing, missing is returned.

recode(x,x1,x2,...,xn) returns missing if x is missing, x1 if x<=x1, x2 if x<=x2, ..., or xn if is greater than x1, x2, ... , x(n-1).

sign(x) returns missing if x is missing, -1 if x<0, 0 if x==0, and 1 if x>0.

sum(x) returns the running sum of x, treating missing values as zero. For example, following the command

generate y = sum(x)

the i-th observation on y contains the sum of the first through i-th observations on x.

System Variables (_variables)

Expressions may also contain _variables (pronounced "underscore variables"). These are built-in, system variables that are created and updated by STATA. They are called _variables because their names all begin with the underscore character (_).

The _variables in STATA are:

_coef[varname] (synonym: **_b**[varname]) contains the value (to machine precision) of the coefficient on varname from the last most recent regression.

_cons is always equal to the number **1**.

_n contains the number of the current observation.

_N contains the total number of observations in the data set.

_pi contains the value of pi to machine precision.

_pred contains the predicted values of the dependent variable from the most recent regression. The predictions are formed using the current values of the regressors which may not be the same as the values they contained when the regression was run. As a result, **_pred** can be used to calculate forecasts and other out-of-sample predictions. For instance, you may **use** one data set, run a regression, then **use** another data set and make predictions using **_pred**.

Elements

_rc contains the value of the return code from the most recent **capture** command.

Explicit Subscripting

Individual observations on variables can be referenced by subscripting the variables. Explicit subscripts are specified by following a variable name with square brackets that contain an expression. The result of the subscript expression is truncated to an integer and the value of the variable for the indicated observation is returned. If the value of the subscript expression is less than **1** or greater than **_N**, a missing value is returned.

As an example, the lagged value of a variable x can be **generate**d by

generate xlag = x[_n-1]

The first observation on xlag is equal to missing value.

When a command is preceded by the **by** varlist: prefix, subscript expressions and the _variables **_n** and **_N** are evaluated relative to the subset of the data currently being processed. For example, in the data set

	bvar	oldvar
1.	1	1.1
2.	1	2.1
3.	1	3.1
4.	2	4.1
5.	2	5.1

the command

by bvar: gen newvar=oldvar[1]

will produce

	bvar	oldvar	newvar
1.	1	1.1	1.1
2.	1	2.1	1.1
3.	1	3.1	1.1
4.	2	4.1	4.1
5.	2	5.1	4.1

Label Values

You may use labels in an expression in place of the numeric values with which they are associated. To use a label in this way, type the label in double quotes followed by a colon and the name of the value label. For instance, if the value label yesno associates the label "yes" with **1** and the label "no" with **0**, then **"yes":yesno** is evaluated as **1**. If the double quoted label is not defined in the indicated value label, or the value label itself is not found, a missing value is returned. Thus, the expression **"maybe":yesno** is evaluated as a missing value.