29

Advanced Macros

- Advanced Macros Overview, 29-2
- What a Local Macro Can Do, 29-4
- Creating and Invoking a Local Macro, 29-5
- The Structure of a Local Macro, 29-6
- Simple Example of a Local Macro, 29-7
- Using Variables, 29-9
- Writing a Template, 29-10
- Declaring Variables, 29-12
- Passing Arguments to Local Macros, 29-13
- Adding Subcommands, 29-14
- Using Text Data in Macros, 29-17
- Specifying a Range with Suffixed Variables, 29-18
- Free Variables, 29-22
- What Next?, 29-26

To Table of Contents  To Index
Advanced Macros Overview

This chapter discusses concepts and techniques for writing advanced macros. If you are new to writing macros, you may want to start out by reading Chapter 28, Creating a Simple Macro.

Terminology: three types of macros

In MINITAB’s documentation, you may see the following terms which distinguish between the three types of MINITAB macros:

- “Global macros” refers to the simplest form of macro, the type discussed in Chapter 28, Creating a Simple Macro.
- “Local macros” refers to the more sophisticated form of macro, discussed in this chapter.
- “Execs” refers to an older form of MINITAB macro, described in Chapter 33, Using Execs.
- “%macros” refers to both global and local macros. Because they share many qualities—for example, both are invoked by typing %, end in the extension MAC, and can use many of the same macro statements—the two types are often discussed together.

If you have Execs from previous releases

If you have Execs that were written using previous releases of MINITAB, you may continue to use them with no change. If you would like to convert them to the new form, it is very easy to do; see Converting Execs to %Macros on page 33-3. If you are writing a new macro, we recommend you write it as a %macro, because we may phase out Execs in a future release of MINITAB and because the new macros provide much greater power and flexibility than do Execs.

The difference between global and local macros

There are two basic types of %macros: global macros and local macros. Both types allow you to create a program of MINITAB commands, to use control statements such as DO-loops and IF statements, and to include subroutines. Both types also allow you to invoke other macros from within a macro. So how do you decide which type you want to use?

When to use global macros

Global macros are usually simpler, and thus easier to write than local macros. Global macros act directly on your current worksheet. When you write a global macro, you
must know which columns, constants, and matrices that will be used when the macro is invoked. For example, you must know that C1 will contain the data, that K2 will contain the correct constant, and that C5 will be an empty column you can use to store the results of the macro.

Global macros, however, cannot invoke local macros, which means that some of the commands available from the menus and dialog boxes will not be available to you. In a number of MINITAB’s dialog boxes, when you click OK, you are actually invoking a local macro. For example, the Stat ➤ Time Series ➤ Trend Analysis dialog box invokes the %TREND local macro. You can see which dialog boxes invoke local macros by looking in the History window after the command has executed; look for a % symbol in front of the command name.

In summary, write a global macro when

■ the task is fairly simple
■ it is possible to know the state of the worksheet ahead of time
■ the task does not require commands that are %macros

When to use local macros

Local macros are more complex than global macros, and thus harder to write, but they are more powerful and flexible. Local macros can use arguments, subcommands, and “local” variables (see What a Local Macro Can Do below). If you need to write a fairly complex macro, or if you want a macro which you can execute like a MINITAB command, then you should write a local macro.

Terminology: two types of worksheets

■ The “global worksheet,” sometimes called the “regular worksheet,” is whatever worksheet is current when you invoke the global macro. The global worksheet consists of more than just the columns of data you see in the Data window—it is all the columns, constants, and matrices you see listed in the Info window for that worksheet. Global macros act directly on the global worksheet.

■ The “local worksheet” is created when you invoke the macro, and is deleted from your computer's memory when the macro finishes. The local worksheet is completely separate from the global worksheet, and is not visible in a Data window. Only the macro can “see” and manipulate the variables in that worksheet—which is why the worksheet is said to be “local” to the macro. You can write your macro to use arguments, so that you can pass variables from the global worksheet to the local worksheet when you invoke the macro, and pass variables out of the local worksheet into the global worksheet when the macro finishes.
What a Local Macro Can Do

Local macros can use arguments

Suppose that you want a macro that will draw a scatter plot with a fitted regression line and 95% confidence bands. With a global macro, you would have to know ahead of time which three columns will contain the data. With a local macro, however, you could specify which columns to use when you invoke the macro.

Arguments can also be used to tell the macro the name of a file to open, the title of a graph, or the number of times to repeat some action. Arguments can also tell the local macro where to store results when it's done. For information on arguments, see Passing Arguments to Local Macros on page 29-13.

Local macros can use subcommands

Local macros can have subcommands that can modify the behavior of the macro—just as subcommands in interactive MINITAB can change the behavior of a command. Subcommands can have their own arguments. You can also choose to include or not include the subcommand when invoking the macro.

For example, the scatter plot macro described above could be made more flexible by including a subcommand that let you decide at what level the confidence bands should be drawn.

<table>
<thead>
<tr>
<th>Type of macro</th>
<th>Invoked by</th>
<th>Does this</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>%REGRPLOT</td>
<td>Draws 95% confidence bands. Worksheet columns to be used for the plots are “hard-coded” in the macro—for example, the data must always be in C1, C2, and C3.</td>
</tr>
<tr>
<td>Local</td>
<td>%REGRPLOT  C15 C22 C34</td>
<td>Draws 95% confidence bands using the worksheet columns you list.</td>
</tr>
</tbody>
</table>

Arguments can also be used to tell the macro the name of a file to open, the title of a graph, or the number of times to repeat some action. Arguments can also tell the local macro where to store results when it's done. For information on arguments, see Passing Arguments to Local Macros on page 29-13.
Local macros can use temporary “local” variables

Local macros also allow you to use temporary variables that are known only to the macro, and exist only while the macro is running. Why does this matter? In interactive MINITAB or in a global macro, the only way you can store results is by storing them in the global worksheet as columns, stored constants, or matrices. This can clutter your worksheet, especially if you need a lot of scratch storage.

With local macros, you can store data in variables and manipulate them as you wish, without affecting your regular worksheet at all. When you exit the local macro, the local variables disappear.

Of course, you may want to change variables in your global worksheet by manipulating them in the local macro. In that case, you can pass those variables in to the local macro as arguments. After the macro executes, the values of those variables will be passed back to the regular worksheet.

Creating and Invoking a Local Macro

Creating a local macro

Local macros are created in the same way as global macros (see Creating a Macro on page 28-4), with only the contents being different. Local macros follow the structure outlined on page 29-6.
To invoke a local macro

1. From a command prompt, enter the symbol % followed by the macro file name, as in %TRIM

2. After the file name, type any arguments that appear on the main command:
   - Unnamed columns, constants, and matrices are not surrounded by quotes, as in %TRIM C1 K2
   - Named columns, constants, and matrices are surrounded by single quotes, as in %TRIM 'Sales' 'NewMean'
   - Text strings, such as titles or file names, are surrounded by double quotes, as in %TRIM C1 K2;
     TITLE "Results";
     STOREIN "OUTPUT.TXT".

3. If the macro has subcommands, type them as in interactive MINITAB, ending each line with a semicolon or a period, as in %TRIM C1 K2;
   PERCENT 4.

4. The default file name extension for macros is MAC. When you invoke a macro that has an extension of MAC, you only need to type the file name, as in %ANALYZE. If the extension is not MAC, you must type the file name and extension, as in %ANALYZE.TXT.

5. When you invoke a macro, by default MINITAB looks for that macro file first in the current directory, then in the \MACROS subdirectory.

6. If the macro is not in one of those default directories, you can specify the directory by including a path when you invoke the macro. For example, %C:\SALES\ANALYZE

7. If a file name includes spaces, put the name in single quotes, as in %'a very long file name.MAC

The Structure of a Local Macro

The contents of a local macro follow this structure:

```
MACRO
template
declaration statements
body of the macro
ENDMACRO
```

The structure of a local macro is similar to that of a global macro, but it includes additional elements that allow you to define the syntax of the user command and to declare variables for the local worksheet.
MACRO and ENDMACRO

MACRO and ENDMACRO mark the beginning and end of each macro. You can have more than one macro in a file—see Invoking Macros from Within Macros on page 30-7. MACRO must be the first line. MACRO (as opposed to GMACRO) says that this is a local macro, not a global macro. MACRO and ENDMACRO can not be abbreviated.

Template

The template gives the macro command name and any subcommands, as well as any arguments. See Writing a Template on page 29-10.

Declaration statements

Each variable that will be used in the macro must be “declared” with a declaration statement. Declaring a variable tells the local macro what type of variable to expect when the macro is invoked: a column, constant, or matrix. See Declaring Variables on page 29-12.

Body of the macro

The body of the macro consists of MINITAB commands, macro statements (such as IF, PAUSE, GOTO, and CALL), and invocations of other local macros. Some MINITAB commands work differently in macros than they do in interactive MINITAB—for a list, see MINITAB Commands that Work Differently in Macros on page 32-7.

Simple Example of a Local Macro

The macro TRIM calculates a 10% trimmed mean—5% trimmed from each end of the data—for a column of data from the global worksheet and stores it in a constant in the global worksheet.

(1) MACRO
(2) TRIM X XBAR
#
# TRIM takes one column, X, as input. It orders the data, trims 5% # from each end, calculates the mean of the remaining data, and # stores it in the constant XBAR.
#
Here is what each line in the macro means:

(1) MACRO marks the beginning of a local macro.

(2) Template. Says to invoke this macro with two arguments: argument 1 is the column of data to be trimmed, and argument 2 is the constant where the trimmed mean is to be stored. See Writing a Template on page 29-10.

(3) Declaration statements.

MCONSTANT declares four constants (N, T1, T2, and XBAR) to be used as variables by the local macro. One of these constants, XBAR, is an argument which corresponds to the constant that is passed into the macro when the user invokes the macro.

MCOLUMN declares three columns (X, XSORT, and XTRIM) to be used as variables by the local macro. One of these columns, X, is an argument which corresponds to the column that is passed into the macro when the user invokes the macro.

See Declaring Variables on page 29-12.

(4) Body of the macro.

(5) ENDMACRO marks the end of the macro.

All lines beginning with the comment symbol # are comments, which are ignored by MINITAB. See Adding Comments on page 28-6.
Invoking TRIM

Suppose you save these lines in a text file called TRIM.MAC stored in your current directory. Now suppose you have data in C5, and you want to calculate the trimmed mean and store it in K1. To invoke the macro, you type,

```
%TRIM C5 K1
``` 

Using Variables

A variable is an alias that can refer to some piece of data: a number, text string, column, constant, or matrix. For example, a variable named “Test1” could represent any of the following: a column of test scores, a constant that is the mean of the test scores, a text string that is the name of the test, etc.

In local macros, use variables like this:

1. On the template, name any variables that will be used as arguments. See Writing a Template on page 29-10.
2. In the declaration statements, define the variable types that were used as arguments. Also declare any other variables that will be used in the macro. See Declaring Variables on page 29-12.
3. In the body of the macro, refer to the variable using its full name. Do not put quotes around the name, as you would in a global macro.

Rules for naming variables

A variable name…

- can be a maximum of eight characters
- may include letters, numbers, and the underscore, but they must begin with a letter
- can be in capitals, lower case, or mixed. On output, variable names appear the way they are written in declaration statements.
- cannot be the same name as a subcommand

For example, some legal variable names are Score, Result1, Result2, My_Test.
Special variables

There are four special-purpose variables that are explained in their own sections:

<table>
<thead>
<tr>
<th>Variable type</th>
<th>Declare with</th>
<th>Contents</th>
<th>For more information see</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subcommand</td>
<td>Do not declare</td>
<td>An implicit constant that has a value of either 1 (if the subcommand was invoked) or 0 (if the subcommand was not invoked)</td>
<td>Checking to see if the subcommand was invoked on page 29-16</td>
</tr>
<tr>
<td>Text</td>
<td>MCONSTANT</td>
<td>A text constant that contains a text string</td>
<td>Using Text Data in Macros on page 29-17</td>
</tr>
<tr>
<td>Suffixed</td>
<td>MCOLUMN MCONSTANT</td>
<td>A range of columns or constants</td>
<td>Specifying a Range with Suffixed Variables on page 29-18</td>
</tr>
<tr>
<td>Free</td>
<td>MFREE</td>
<td>Column, constant, or matrix whose type is undetermined until the macro is invoked</td>
<td>Free Variables on page 29-22</td>
</tr>
</tbody>
</table>

Writing a Template

The template gives the macro command name and any subcommands, as well as any arguments. Arguments are variables that appear on the macro command and subcommand lines that are passed into and out of the macro.

For more information on arguments, see Passing Arguments to Local Macros on page 29-13. For information on subcommands, see Adding Subcommands on page 29-14.
Writing a Template

Advanced Macros

Syntax

- The only lines that can appear between the word MACRO and the template are comment lines that begin with #.
- The first line of the template contains the macro name. It is good practice to use the same name for the template and the file name, but it is not necessary to do so. The file name is used when you invoke the macro.
- Command and subcommand names can contain letters, numbers, and the underscore character, to a maximum of eight characters. They must start with a letter. Only the first four letters of macro subcommands are used by MINITAB.
- Arguments must have legal variable names. See Rules for naming variables on page 29-9.
- You may have two or more macros in one file. In that case, each macro follows the structure shown in The Structure of a Local Macro on page 29-6, and each must have a unique template name. When you invoke a macro, MINITAB executes the first macro in the file. Subsequent macros in the file are subroutines that you can invoke using a CALL statement (see page 30-8).
- If the command has subcommands, use punctuation just as in interactive MINITAB: end each line with a semi-colon, and put a period after the last subcommand.

► Example of a template for a command with arguments

**Template**

```
commandname [argument1] [argument2][...]
[subcommandname1 [argument1] [argument2][...]]
[subcommandname2 [argument1] [argument2][...]]
...
```

**Invoked by**

```
commandname
```

In the template, Trim is the command (and name of the macro), X is the first argument, and Xbar is the second argument. The X variable is the column (to be specified when the macro is invoked) where the macro should look for data. Xbar is the constant where the macro should store the result. For more information on arguments, see Passing Arguments to Local Macros on page 29-13.

► Example of a template for a command with a subcommand

**Template**

```
commandname [argument1] [argument2][...]
[subcommandname1 [argument1] [argument2][...]]
[subcommandname2 [argument1] [argument2][...]]
...
```

**Invoked by**

```
commandname
```

In the template, the TRIM command has its arguments x and Xbar. The subcommand is Percent. Percent has an argument, Pct, that can contain a constant.
Declaring Variables

All variables used in a local macro (with the exception of subcommand constants, as noted in Special variables on page 29-10) must be declared. Declaring a variable tells the local macro what type of variable to expect.

Syntax

- Declare variables that are constants with MCONSTANT, variables that are columns with MCOLUMN, and variables that are matrices with MMATRIX. (You may also use the synonyms MCONSTANTS, MCOLUMNS, and MMATRICES.) After the M- command, list all the variables that are of that type, separated by a space. You may use a declaration statement several times.

- An argument (that is, a variable in the template) may be given the declaration MFREE. Then its type—column, constant, or matrix—is determined by the type of the variable that is passed when the macro is invoked. The macro statement MTYPE (page 29-22) allows you to determine whether a variable declared with MFREE is a column, constant, or matrix. For more information, see Free Variables on page 29-22.

- Once a variable is declared, it cannot be redeclared.

- The declaration commands (MCOLUMN, MCONSTANT, etc.) cannot be abbreviated.

- The declared variable must have a legal name. See Rules for naming variables on page 29-9.

For more information on using variables, see Using Variables on page 29-9.

Example of declaring variables

For example, suppose the template is

\texttt{TRIM X Xbar}
Passing Arguments to Local Macros

TRIM is the name of the macro and X and Xbar are variables that will be passed into the macro. The macro would need declaration statements that define whether X and Xbar are constants, columns, matrices, or “free” variables (defined below). Let’s say X is a column in the global worksheet and Xbar is a constant in the global worksheet. The user would invoke the macro by typing, say, %TRIM C5 K1. The first few lines of the local macro file would then be

```
MACRO
TRIM X
MCOLUMN X
MCONSTANT Xbar
```

Note
If you see the error “Missing END for READ, SET, or INSERT,” it may be because you have named a local variable with the same name as a MINITAB command, and entered it after READ, SET, or INSERT. For example:

```
SET col 1
min: max/1
END
```

where min and max are local variable names. MINITAB interprets the second line as a command because MIN and MAX are also MINITAB commands. It displays the error message because it thinks you are trying to execute a command without first having entered the required END statement. You must avoid using MINITAB commands for variable names if you need to use them in this way.

Passing Arguments to Local Macros

Arguments are variables that are passed into and out of a macro when it is invoked. They are listed on the main command line and subcommand lines of the macro. If you pass a global worksheet variable (a column, constant, or matrix) to a macro and the macro changes the value of that variable, the global worksheet variable will contain that changed value after the macro executes.

If an argument has a name in the global worksheet—for example, if the column C1 has a name like ‘Results,’ or if the constant K1 has a name that was assigned using the NAME command—that name is also passed into and out of the macro, along with the argument. For more information on how names are handled in macros, see How MINITAB Labels Output in Local Macros on page 31-7.

Within the macro, you can also change the name of a variable passed in as an argument, then pass the name back out to the global worksheet. For example, the variable K1 could be given the name TestMean within the macro; when the macro finished, K1 would show the name TestMean in the Info window. For details, see Changing the Name of an Argument on page 31-8.

An argument can be

- a stored column, constant, or matrix from the global worksheet, such as ‘Sales’, C1, K2, or M1
Adding Subcommands

Local macros can have subcommands that can modify the behavior of the macro—just as in interactive MINITAB. Subcommands can have their own arguments. You can also choose to include or not include the subcommand when invoking the macro.

To add subcommands to a macro

1. Write a template that includes a subcommand. See Example of a template for a command with a subcommand on page 29-11.

2. If any of your subcommands include arguments that are constants, you can assign default statements to those arguments. See Assigning default values to subcommand arguments that are stored constants on page 29-15.

3. Invoke the macro as you would any other macro, following its template.

Invoking macros that use subcommands

- When invoking a macro, if you type a subcommand more than once, MINITAB uses the first occurrence of the subcommand.

- Individual arguments on subcommands cannot be optional. For example, suppose a subcommand has two arguments. When you invoke the macro, you can either omit the subcommand entirely or use it with two arguments. You cannot use it with one argument and take a default for the other argument.

Example of a macro with a subcommand

Suppose we improve TRIM by adding an optional subcommand, PERCENT, that allows the user to specify the trimming percent. If the user does not specify PERCENT,
we use the default value of 5%. We give this default value using the macro statement DEFAULT. Here is the macro:

```
MACRO
  TRIM2 X XBAR;
  PERCENT PCT;
  #
  # TRIM2 takes one column, X, as input. It orders the data, trims the percent specified by PCT from each end, calculates the mean of the remaining data and stores it in XBAR.
  # If PCT is not given, 5% is used.
  #
  MCONSTANT N T1 T2 XBAR PCT
  MCOLUMN X XSORT XTRIM M
  DEFAULT PCT = 5
  #
  # First we calculate the trimming points T1 and T2.
  LET N = COUNT(X)
  LET T1 = ROUND(N*PCT/100)
  LET T2 = N - T1 + 1
  # Next we check for the case when T1 = 0 and nothing is trimmed.
  IF T1 = 0
    LET XTRIM = X
  # Otherwise, we sort X, trim the ends and calculate the mean.
  ELSE
    LET XSORT = SORT(X)
    COPY XSORT XTRIM M
    OMIT 1:T1 T2:N.
    LET XBAR = MEAN(XTRIM)
  ENDIF
ENDMACRO
```

We will store this program in a file called TRIM2.MAC.

Now suppose, in your global worksheet, you have data in a column named Score and you want to calculate the 4% trimmed mean and store it in a constant named Sbar. Type,

```
%TRIM2 'Score' 'Sbar';
  PERCENT 4.
```

When you invoke a macro, you must use single-quotes around variable names, as with most other MINITAB commands. It is only in the macro text that quotes are not used.

**Assigning default values to subcommand arguments that are stored constants**

The DEFAULT statement is an optional line that allows you to assign a default value to a stored constant that appears on an optional subcommand. If a subcommand is not used when a user invokes the macro, the value on the DEFAULT line is used for the subcommand argument.
You cannot use DEFAULT to assign values to arguments on the main command—only arguments that are stored constants for a subcommand. Defaults for columns and matrices must be handled within the body of the macro.

```
DEFAULT argument1 = value argument2 = value ...
```

Two rules about the syntax of DEFAULT:

- The DEFAULT line must come immediately after the declaration statements, before any other commands in the macro.
- The DEFAULT command cannot be abbreviated.

Let’s say that the template of a local macro specifies a main command and an optional subcommand, as in

```
TRIM2 X XBAR;
PERCENT PCT;
```

and that the default statement (after the declaration statements) is

```
DEFAULT PCT = 5
```

If the user invokes the macro with only %TRIM C1 K1, and does not use the optional PERCENT subcommand, the DEFAULT statement will assign the value 5 to the variable PCT.

**Checking to see if the subcommand was invoked**

As with regular MINITAB commands, subcommands of macros are optional—when users invoke the macro, they can choose whether or not to type the subcommand. You can structure your macro to respond differently depending on whether or not a subcommand was used.

Each subcommand listed on the template is an **implicit constant**, which means that it is automatically created and does not have to be declared. This is why there is a rule against declaring a variable with the same name as a subcommand.

If the macro is invoked using the optional subcommand, MINITAB sets the subcommand constant to 1; if the subcommand was not used, MINITAB sets the subcommand constant to 0.

In the macro below, if the user types the PERCENT subcommand, MINITAB sets the variable PERCENT equal to 1. If the user does not type PERCENT, the variable is set equal to 0. The NOTE command after the “IF PERCENT = 0” statement tells the user when the macro is using the default trim size of 5 percent.

```
MACRO
TRIM2 X XBAR;
PERCENT PCT;
MCONSTANT N T1 T2 XBAR PCT
```
Using Text Data in Macros

In macros you can use text data in columns, in stored constants, and as text strings, just as you can in the rest of MINITAB.

In addition, you can pass a text string into a macro; when you invoke the macro, enclose the string in double quotes. The passed string can be assigned to a constant in your macro. Constants that hold text data are useful for specifying graph titles, file names, and names for variables that will be created in the macro.

Example of passing text strings to specify file names

Here is a simple example.

MACRO
REVERSE file1 file2
#
# REVERSE reads in the first 3 columns of the input file, file1, then
# stores these 3 columns in reverse order in the output file, file2.
#
MCONSTANT file1 file2
MCOLUMN X Y Z
PRINT file1 file2
READ X Y Z
FILE file1
WRITE file2 Z Y X
ENDMACRO

We could use this macro to reverse the columns in the file called INPUT.DAT and store the reversed data in the file called OUTPUT.DAT by using

%REVERSE "INPUT" "OUTPUT"

Note the use of quotes. The file names are text strings, like all other text strings, must be enclosed in double quotes. (In older versions of MINITAB, single quotes were used instead of double quotes. Single quotes still work but are not recommended and can result in ambiguities concerning variable names and the contents of constants.)

Note also that in the global worksheet you can store the file names in constants, and use the stored constants as arguments in the macro. For example:

LET K1 = "INPUT"
LET K2 = "OUTPUT"
%REVERSE K1 K2
Commands for storing text in constants

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KKNAME K C, ..., K C</td>
<td>Stores the name of column C in the constant K.</td>
</tr>
<tr>
<td>KKSET K &quot;text&quot;, ..., K &quot;text&quot;</td>
<td>Stores the text within the double quotes in the constant K.</td>
</tr>
<tr>
<td>KKCAT K K K</td>
<td>Concatenates, or combines, the text in the first constant K with the text in the second constant K, and stores the combined string of text in the third constant K.</td>
</tr>
</tbody>
</table>

These three macro commands allow you to store text in a constant. They are especially useful for displaying titles and other annotation on macro output.

Note

| KKNAME, KKSET, and KKCAT are macro commands, which means that they can be used only in global or local macros. They cannot be used in Execs or while using MINITAB interactively.

KKNAME stores the name of column C in the constant K.

KKSET stores the text within the double quotes in the constant K. You can also use the regular MINITAB command LET to store text in constants. KKSET, however, can store several text strings in several constants at once, whereas LET stores one text string in one constant. (Note, in older versions of MINITAB, you used single quotes around the text in KKSET. You can still use single quotes, but they are not recommended).

KKCAT concatenates, or combines, the text in the first constant K with the text in the second constant K, and stores the combined string of text in the third constant K. For example, if the constant X contained “Mr.” and the text constant Y contained “Jones”, the command

```
KKCAT X Y Z
```

would put the string “Mr.Jones” in the constant Z.

Note

The MINITAB command CONCATENATE combines columns containing text data, whereas the macro command KKCAT combines constants containing text data.

Specifying a Range with Suffixed Variables

A suffixed variable is most useful when

- you want to abbreviate a list of known variables—this is a defined range. For example, if a command in a macro acts on five columns, it is easier to write C1-C5 than C1, C2, C3, C4, C5.
- you do not know until the macro is invoked how long a list will be—this is an undetermined range. For example, the user may want the macro to act on C1-C3, C1-C5, or C1-100.
Syntax

A suffixed variable is a variable name followed by a period followed by the suffix. The suffix can either be an integer or a stored constant. The range of suffixed variables can be abbreviated using a dash.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Period</th>
<th>Suffix</th>
<th>Suffed variable</th>
<th>Range of suffixed variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>.</td>
<td>1</td>
<td>X.1</td>
<td>X.1-X.5</td>
</tr>
<tr>
<td>My_Data</td>
<td>.</td>
<td>1</td>
<td>My_Data.1</td>
<td>My_Data.1-My_Data.5</td>
</tr>
<tr>
<td>Test</td>
<td>.</td>
<td>1</td>
<td>Test.1</td>
<td>Test.1-Test.num</td>
</tr>
<tr>
<td>Test</td>
<td>.</td>
<td>num</td>
<td>Test.num</td>
<td></td>
</tr>
</tbody>
</table>

The variable name and the suffix can each have up to eight characters. With the period, that means a suffixed variable name can have up to 17 characters. However, if a MINITAB command prints out the name of a suffixed variable, only the last eight characters are used. So if you plan to print out suffixed variables, you should probably keep them short, as in Col.1-Col.5, X.1-X.N, etc.

Using suffixed variables in the template and declarations

Within the body of a macro, suffixed variables can be used in any order, alone or in groups. But when they appear on the template or in declaration statements, they must follow these rules:

- In the template and declarations, you must give a list of suffixed variables as one complete list, in order, and using a dash. All variables in the list must be of the same type.

<table>
<thead>
<tr>
<th>Templates</th>
<th>Declarations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legal:</td>
<td></td>
</tr>
<tr>
<td>TRIM X.1-X.5</td>
<td>MCOLUMN X.1-X.5</td>
</tr>
<tr>
<td>TRIM X.1-X.5 Y.1-Y.8</td>
<td>MCONSTANT X.1-X.5 Y.1-Y.8</td>
</tr>
<tr>
<td>TRIM Z X.3-X 20 V1 V2</td>
<td>MCOLUMN Z X.3-X 20 V1 V2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Illegal:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIM X.1-X 3 X.4-X 5</td>
<td>MCOLUMN X.1-X 3 X.4-X 5</td>
</tr>
<tr>
<td>TRIM X.1-X 2 Y X.3-X 5</td>
<td>MCONSTANT X.1-X 2 Y X.3-X 5</td>
</tr>
<tr>
<td>TRIM X 5-X 1</td>
<td>MCOLUMN X 5-X 1</td>
</tr>
</tbody>
</table>
Chapter 29 Specifying a Range with Suffixed Variables

- In the template each command and subcommand can have as many regular arguments and as many defined-range arguments as you wish, but the command or subcommand can have only one undertermined-range argument.

Legal template statements:

\[
\begin{align*}
\text{MPROC1} & \quad X \cdot 1 - X \cdot 10 \quad Y \cdot 1 - Y \cdot N \\
\text{MPROC2} & \quad X \cdot 1 - X \cdot 10 \quad Y \cdot 4 - Y \cdot 20 \\
\text{MPROC3} & \quad X \cdot 1 - X \cdot M \\
& \quad \text{SUB1} \quad Y \cdot 1 - Y \cdot N \\
& \quad \text{SUB2} \quad Z \cdot 5 - Z \cdot P \quad W \cdot 1 - W \cdot 10.
\end{align*}
\]

Illegal template statement: \text{MPROC4} \quad X \cdot 1 - X \cdot M \cdot Y \cdot 1 - Y \cdot N

- Once you have declared a suffixed variable, you cannot declare another variable, even one of the same type, with the same prefix. The following two declarations cannot be used in the same program. Because the prefix “X” is used with MCOLUMN, it cannot be used again—either for additional columns or for any other type of variable.

Legal first declaration: \text{MCOLUMN} \quad X \cdot 1 - X \cdot 5

Illegal second declaration: \text{MCOLUMN} \quad X \cdot 6 - X \cdot 10

- Do not declare the suffix of a suffixed variable. For example, suppose you have the range X.1-X.N. You do not give N a value; MINITAB does this when you invoke the command.

Example of suffixed variables with a defined range

The macro GENMEDIANS generates five columns of random data, then stores the median of each row in another column.

\[
\begin{align*}
\text{MACRO} \\
\text{GENMEDANS MEDIANS} \\
\# \\
\text{MCOLUMN} \quad X \cdot 1 - X \cdot 5 \quad \text{MEDANS} \\
\# \\
\text{RANDOM} \quad 100 \quad X \cdot 1 - X \cdot 5 \\
\text{RMEDAN} \quad X \cdot 1 - X \cdot 5 \quad \text{MEDANS} \\
\text{ENDMACRO}
\end{align*}
\]

There is one list of 5 columns, X.1, X.2, X.3, X.4, X.5, and a single column, MEDIANS. The variables in a list are always stored together in the worksheet. Notice that a dash abbreviates this list.

Suppose you stored this macro in a file called GEN2.MAC, and invoke it with

\%
\text{GENMEDANS C10}
\%

After the macro finishes, the medians would appear in C10.
Example of using a constant to define a range of columns

The following modification, called GEN2, allows the user to use the subcommand OBS to specify the number of observations in each sample (M).

MACRO
GEN2 MEDIANS;
OBS M.
#
MCOLUMN X.1-X.M MEDIANS
MCONSTANT M
DEFAULT M = 5
#
RANDOM 100 X.1-X.M
RMED AN X.1-X.M MEDIANS
ENDMACRO

Suppose you stored this macro in a file called GEN2.MAC, and invoke it with

%GEN2 C1;
OBS 10.

This generates 100 rows in the local worksheet, each containing 10 observations stored in X.1-X.10. The median of each row is calculated and stored in the macro variable MEDIANS. When the macro finishes, the medians appear in the column C1.

Example of suffixed variables with an undetermined range

The following macro, ORSTATS, takes a list of columns and calculates three rowwise order statistics, the minimum, median, and maximum.

MACRO
ORSTATS X.1-X.N MIN MED MAX
#
# Input consists of a list of columns X.1-X.N.
# The rowwise minimums, medians, and maximums are calculated and
# stored in MIN, MED, and MAX respectively.
#
MCOLUMN X.1-X.N MIN MED MAX
#
RMIN X.1-X.N MIN
RMED X.1-X.N MED
RMAX X.1-X.N MAX
ENDMACRO

Suppose we want to calculate the same statistics for eight columns, C5–C12, and store them in C21, C22, and C23. When invoking the macro, we would type

%ORSTATS C5-C12 C21-C23

By matching arguments on this line with the template in the macro program, MINITAB determines that N = 8. Then MINITAB matches C5 to X.1, C6 to X.2, ..., C12 to X.8 and C21 to MIN, C22 to MED, and C23 to MAX.
Free Variables

You may want a macro that will operate on a column, constant, or matrix—whatever the user decides to use when he or she invokes the macro. The macro can then take appropriate action, depending on the type of variable used when invoking the macro. A free variable is an argument whose type—column, constant, or matrix—is not determined until the macro is invoked.

To use a free variable in a macro

You must do five things in the code of your macro to make free variables work:

1. List the free variable as an argument on the template. For example, here is a template for the macro TELLME that has X as an argument:
   
   TELLME X

2. Declare the free variable with the declaration statement MFREE. For example:
   
   MFREE X

3. Declare another variable as a constant:
   
   MCONSTANT Vartype

4. Use the macro statement MTYPE to look at the free variable and store its variable type number in the constant declared in step 3. The syntax is

   MTYPE the variable type of variablename is stored in K

   If the variable is a constant, then K is set to 1; if it is a column, K is set to 2; and if it is a matrix, K is set to 3. You can include an MTYPE statement anywhere within the body of a local macro.

   For example,

   MTYPE X Vartype

   looks at the free variable X and stores its variable type (1, 2, or 3) in the constant Vartype. If the free variable is a constant, then Vartype is set to 1; if it is a column, Vartype is set to 2; and if it is a matrix, Vartype is set to 3.

5. Write code that can respond to the variable type that was used. In the following example, the IF statements make the macro perform different actions depending on what type of variable X is:

   IF Vartype = 1
   NOTE X is a constant!
   ELSEIF Vartype = 2
   NOTE X is a column!
   ELSE
   NOTE X is a matrix!
   END IF

6. Invoke it. Macros that use free variables are invoked just like any other local macro (for details, see Creating and Invoking a Local Macro on page 29-5).
**Free Variables**

**Advanced Macros**

---

**Note**

There is one case when the macro processor cannot determine the type of a variable. This happens when a variable that appears on an optional subcommand is declared as MFREE, and a user invokes the macro without using the subcommand. In this case, the macro processor assumes the variable is a column.

**Example of a macro that uses free variables**

The macro TELLME performs the fairly obvious task of telling a user what kind of variable was used when the variable was invoked. Here is the complete code:

```plaintext
MACRO
    TELLME X
    MFREE X
    MCONSTANT Vartype
    MTYPE X Vartype
    IF Vartype = 1
        NOTE X is a constant!
    ELSEIF Vartype = 2
        NOTE X is a column!
    ELSE
        NOTE X is a matrix!
    END F
ENDMACRO
```

TELLME can be invoked in all of the following ways, and will produce the following output in the Session window:

<table>
<thead>
<tr>
<th>Invoked like this</th>
<th>Produces this</th>
</tr>
</thead>
<tbody>
<tr>
<td>%TELLME C1</td>
<td>X is a column!</td>
</tr>
<tr>
<td>%TELLME K1</td>
<td>X is a constant!</td>
</tr>
<tr>
<td>%TELLME M1</td>
<td>X is a matrix!</td>
</tr>
</tbody>
</table>

**Example of a more complex macro that uses free variables**

In the following macro BETWEEN.MAC, the arguments LOW and HI can be either columns or constants.

```plaintext
MACRO
    BETWEEN X 1-X N LOW H ANS;
    STRICT CT.
    MCOLUMN X 1-X N L H ANS
    MFREE LOW H
    #
    # X 1-X N is a list of columns. LOW and H can each be either
    # a column or a constant.
    #
    # BETWEEN checks to see if the values in one row of X 1-X N are
    # all greater than or equal to LOW and all less than or equal
    # to H. If they are, the corresponding row of ANS is set 1.
    # If not then ANS is set to 0. If the STRICT subcommand is used
    # then BETWEEN checks for < and > rather than <= and >=
    #
    ENDMACRO
```

---

MINITAB User's Guide 1 29-23
DTYPE: Finding the Data Type of a Variable

Use DTYPE if you need to find out the data type of a column or constant. You can find out if the column or constant contains text, numeric, or date/time data, or whether the column or constant contains no data at all. If the data are numeric, DTYPE can also tell you if the data are real numbers or integers.

DTYPE is very useful when parts of your macro only work on some types of data. For example, you may have a subcommand of your local macro that lets the user specify a title for a graph; DTYPE can tell you if the user specified a text string or a number. Or, perhaps a part of your macro requires an integer; DTYPE could tell you if a variable was not an integer, allowing your macro to convert the real number to an integer.

The syntax for DTYPE is as follows:

```
DTYPE of variablename is stored in K
```
**Variablename** is the name of a constant or column.

*K* is the constant where you want the DTYPE code to be stored. The possible DTYPE codes are as follows:

<table>
<thead>
<tr>
<th>DTYPE code</th>
<th>Means column or constant is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Text</td>
</tr>
<tr>
<td>1</td>
<td>Real number</td>
</tr>
<tr>
<td>2</td>
<td>Integer</td>
</tr>
<tr>
<td>3</td>
<td>Date/time</td>
</tr>
<tr>
<td>10</td>
<td>Empty</td>
</tr>
</tbody>
</table>

**Example of a macro that uses DTYPE**

The macro TELLDATA is a variation of the TELLME macro listed under *Example of a macro that uses free variables* on page 29-23. TELLDATA tells a user the data type of the variable specified when the macro is invoked. Here is the complete code:

```plaintext
MACRO
    TELLDATA X
    MFREE X
    MCONSTANT Vartype
    DTYPE X Vartype
    IF Vartype = 0
    NOTE Variable is text
    ELSEIF Vartype = 1
    NOTE Variable is real number
    ELSEIF Vartype = 2
    NOTE Variable is integer
    ELSEIF Vartype = 3
    NOTE Variable is date/time
    ELSEIF Vartype = 10
    NOTE Variable is empty
    END IF
    ENDMACRO
```

Say that you have a worksheet that contains the following variables:

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>K1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>1.5</td>
<td>Hello</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sally</td>
<td>3.5</td>
<td></td>
</tr>
</tbody>
</table>
TELLDATA can be invoked in all of the following ways, and will produce the following output in the Session window:

<table>
<thead>
<tr>
<th>Invoked like this</th>
<th>Produces this</th>
</tr>
</thead>
<tbody>
<tr>
<td>%TELLDATA C1</td>
<td>Variable is integer</td>
</tr>
<tr>
<td>%TELLDATA “Hello”</td>
<td>Variable is text</td>
</tr>
<tr>
<td>%TELLDATA K1</td>
<td>Variable is text</td>
</tr>
<tr>
<td>%TELLDATA C2</td>
<td>Variable is text</td>
</tr>
<tr>
<td>%TELLDATA C3</td>
<td>Variable is real number</td>
</tr>
</tbody>
</table>

**What Next?**

Look in these chapters for more information on writing macros:

- Chapter 30, *Controlling Macro Flow*, describes techniques and commands you can use to control which commands are executed, and when.
- Chapter 31, *Managing Input and Output*, shows you how to make a macro interactive, label output, save data, and more.
- Chapter 32, *Handling Errors in Macros*, discusses how to interpret error messages, which MINITAB commands behave differently in macros, and tools you can use to track down and correct problems in macros.
- Chapter 33, *Using Execs*, discusses MINITAB’s older macro functionality, called Execs.