Minitab®

Minitab Macros Help

Minitab®

Contents

Overview

A Minitab macro is a file that contains a set of session commands. You can use a Minitab macro to automate a repetitive task, such as generating a monthly report, or to extend Minitab's functionality, such as computing a special test statistic.

Write a macro

1. Type the macro in a text editor. You can also copy commands from the **History** pane to repeat actions that you already completed.
2. Save and name the macro.

Run the macro

1. Open the **Command Line** pane and type % followed by the macro name, for example, %mymacro.
2. Click **Run**

Updates for release 19.1

The following section describes changes to the behavior of Minitab macros.

Input and history

In Minitab 19.1, choose **View** > **Command Line/History** to open the **Command Line** pane and the **History** pane. In the **Command Line** pane, enter session commands and run macros. Use the **History** pane to see session commands that ran and to copy those session commands.

Obsolete continuation character (&)

In previous versions the & symbol indicated that a command continued on the next line, for example:

```
PLS C18 = C1-C17 c1*c2 c1*c3 c1*c4 c1*c5 c1*c6 c1*c7 c1*c8 c1*c9 c1*c10 c1*c11&
 c1*c12 c1*c13 c1*c14 c1*c15 c1*c16 c1*c17;
```

In Minitab 19.1, session commands with an & symbol create errors. Instead, type everything on 1 line.

```
PLS C18 = C1-C17 c1*c2 c1*c3 c1*c4 c1*c5 c1*c6 c1*c7 c1*c8 c1*c9 c1*c10 c1*c11 c1*c12
 c1*c13 c1*c14 c1*c15 c1*c16 c1*c17;
```

Adding comments and notes

In Minitab 19.1, each instance of the `NOTE` command creates a new output tab. To keep different notes together, surround all of the notes and output that you want on one output tab with `MTITLE` and `ENDMTITLE`. For more information, go to Adding Comments and Notes on page 13.

Obsolete commands

The following commands do not function in Minitab 19.1 and had specific uses in macros. For a complete list of changes to session commands, run `HELP` in the **Command Line** pane to open Session Commands Help.

DIR
List the names of files in a directory.

GPAUSE
Specify the number of graphs to display before you are prompted to save or discard open graphs. In Minitab 19.1, the number of graphs does not have a fixed limit.

GPRINT
Print a graph window. In Minitab 19.1, all output is in tabs instead of windows.

INSERT
Insert rows of data into the worksheet. Consider `WOPEN` and `READ`.

MRESET
Restore environment settings to pre-macro conditions. In Minitab 19.1, restoration occurs at the end of every macro.

PLUG/NOPLUG
Respond to errors from the macro processor. In Minitab 19.1, the macro processor stops when it encounters an error.

`TITLE/NOTITLE`

Display a title above session window output. In Minitab 19.1, use `MTITLE`/`ENDMTITLE` to add a title for an output tab and to group output on a single output tab.

`TYPE`

Display the text of a standard ASCII file.

`YES/NO`

Set a constant to yes or no depending on the response of Y or N. Use the `TERMINAL` subcommand to get a response from the keyboard and the `IF` command to set a value with the response. For more information on getting a response, go to READ, TSET, and SET: Session command for asking users questions and using the answers in a macro on page 54. For more information on the IF command, go to IF, ELSEIF, ELSE, ENDIF: Session commands for executing code depending on a logical condition on page 46.

Commands by function

In addition to all the session commands, macros also have exclusive commands that assist in processing the macro.

Some session commands cannot be used in macros. For more information, go to Commands and subcommands that are not allowed in macros on page 29.

Structure commands

GMACRO, MACRO, and ENDMACRO: Session commands for marking the beginning and ending of a macro on page 43

Declaration statement commands

MCONSTANT, MCOLUMN, MMATRIX, and MTYPE: Session commands for declaring variables on page 49

MFREE: Session command for declaring a free variable on page 50

DEFAULT: Session command for assigning default values to subcommand arguments on page 40

Local macro variable commands

KKCAT, KKNAME, and KKSET: Session commands for using text on page 47

DTYPE: Session command for determining the data type of a column or a constant on page 41

Control statement commands

IF, ELSEIF, ELSE, ENDIF: Session commands for executing code depending on a logical condition on page 46

DO and ENDDO: Session commands for looping through a block of commands on page 40

WHILE and ENDWHILE: Session commands for repeating a block of commands depending on a logical expression on page 55

NEXT: Session command for transferring control from a loop to the beginning of the block on page 52

BREAK: Session command for transferring control from a DO- or WHILE-loop on page 36

GOTO and MLABEL: Session commands for branching to any line in a macro on page 44

CALL and RETURN: Session commands for passing control to another macro on page 38

EXIT: Session command for transferring control back to Minitab or for closing Minitab on page 42

PAUSE and RESUME: Session commands for pausing and resuming a macro on page 53

DOS commands

CD: Session command for displaying or changing the current directory on page 39

Commands for labeling output

MTITLE: Session command for adding a title above output on page 52

WTITLE: Session subcommand for specifying the title of the output pane on page 56

Debugging commands

Commands for error handling

Other local macro commands

Commands that affect output

Commands for communicating with macro users

Exec commands

Using macros

Introduction, Simple Macros

Macros Terms and Overview

Terminology: three types of macros

Three types of macros have been developed in Minitab to perform various repetitive tasks easily and effectively. In Minitab's documentation, you may see the following terms which distinguish between the three types of Minitab macros:

- **Global macros**, also referred to as simple macros. Global macros let you run Minitab command language and some control statements, such as IF statements and loops.
- **Local macros**, also referred to as advanced macros. Local macros add functionality that global macros do not have, such as the ability to take subcommands and to make use of variables that are not stored in the worksheet.
- **Execs** are the simplest macros. Execs run only Minitab command language.

Using Global Macros on page 11

Using Local Macros on page 14

Using Execs on page 30

Note For global macros and local macros, the decimal separator is always a period (.) and the list separator is always a comma (,). For execs, the decimal separator and list separator match the operating system and the syntax of Minitab session commands.

Similarities between Local and Global Macros

Because global and local macros 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. Both global and local macros 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.

Terminology: two types of worksheets

Worksheets include all the data that are contained in a project. While most menu and session commands use only one worksheet, macros use two different types of worksheets.

1. Global worksheet: Both local and global macros work with a global worksheet.
2. Local worksheet: Only local macros work with a local worksheet.

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 pane - it is all the columns, constants, and matrices associated with the worksheet. To see this other information, right-click on the worksheet tab at the bottom of the worksheet and select Worksheet Information. Global macros act directly on the global worksheet.

The local worksheet is created when you invoke the macro. The local worksheet 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 pane. 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.

Global Macro Structure

A macro consists of lines of text, which represent command language, stored in a text file. While all macros follow a similar structure, global macros follow this specific structure:

```
GMACRO
template
body of the macro
ENDMACRO
```

GMACRO and ENDMACRO

These commands mark the beginning and end of each global macro. `GMACRO` must be the first line of your macro because it labels the macro type as global, not local. `ENDMACRO` ends the macro command. `GMACRO` and `ENDMACRO`, as well as all macro commands, cannot be abbreviated.

Template

The term "template" is used much differently when discussing global macros than when discussing local macros. Global macros simply use a "template" to name the group of commands for the macro. Local macros use a "template" to store the most repetitive commands, subcommands, and corresponding arguments.

You type the name of the template for your global macro starting with a letter. The remaining characters in the name can contain letters, numbers, or the underscore character. The template name can be upper, lower, or mixed case; Minitab ignores case when you invoke the macro. Using the macro file name as your template name is probably most convenient, but not required. For example, all the following are valid combinations of templates and file names.

Template	**File name**	**Invoked by**
MyMacro	MYMACRO.MAC	`%MYMACRO`
Analyze	TEST.MAC	`%TEST`
Analyze2	TEST2.TXT	`%TEST2.TXT`

Body of the macro

The body of a macro consists of command language that controls the automatic data processing. The language includes:

- Minitab commands
- Control statements
- Macro statements (such as `IF`, `THEN`, `PAUSE`, `CALL` and `GOTO`)
- Invocation of other global macros

Creating a Global Macro

To create a global macro using a text application

1. Write your macro using any text editor, such as Notepad.

2. Save the updated global macro file in text-only format, with a file name and the file extension .MAC. If you save the file to one of the following folders, then you do not have to specify the file path when you run the macro:
   - The folder where the project is
   - The **Default file location** for Minitab
   - The **Macro location** for Minitab
   - The Macros subfolder of the main Minitab folder

   To specify the **Default file location** or the **Macro location**, choose **File** > **Options**, select **General**.

To create a global macro using Minitab

1. Execute a series of commands using either menu commands or typing session commands into the **Command Line** pane.
2. Click in the **History** pane. This pane displays all the commands executed in your session.
3. Highlight the commands you want to include in your macro, right click on them and choose **Copy**.
4. **Paste** into any text editor, such as Notepad.
5. Change any commands if you wish. Then insert three lines to include `GMACRO`, the template and `ENDMACRO`.
6. Save the updated global macro file in text-only format, with a file name and the file extension .MAC. If you save the file to the folder that is the macro location in Minitab, then you can invoke macros without specifying the file path. In Windows, the default macro location is the Documents folder for the current user. To specify the location for macros, choose **File** > **Options**, select **General**, and specify the folder in **Macro location**.

Example of a Global Macro

Here is a simple example of a macro file named ANALYZE.MAC. Indenting is not necessary, but may be done to improve readability as illustrated here. The macro creates random data in C1-C3. Then, the macro proceeds with a regression analysis on the natural log of the variable in c3.

`GMACRO`	Marks the beginning of the global macro.
`Analyze`	The template, or the name, of this macro.

`MTITLE "Analysis of Yield"`
`NAME C1 "Yield" C2 "Chem1" C3 "Chem2" C5 "Ln.Yield"`
`RAND 50 C1-C3;`
`UNIFORM 5 1.5.`
`PRINT C1-C3`
`DESCRIBE C1-C3`
`LET C5 = LOGE('Yield')`
`REGRESS;`
` RESPONSE C5;`
` CONTINUOUS C2 C3;`
` TERMS C2 C3.`
`ENDMTITLE`
	Body of the macro.

`ENDMACRO`	Marks the end of the macro.

Invoking a Global Macro

To invoke, or process, a global macro from Minitab, enter the symbol % followed by the macro file name. For example, to invoke a macro file named ANALYZE.MAC, enter the command: `%ANALYZE`

Notes on invoking macros

- 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`.
- If you invoke a macro that is one of the following folders, then you do not have to specify the file path when you run the macro:
  - The folder where the project is
  - The **Default file location** for Minitab
  - The **Macro location** for Minitab
  - The Macros subfolder of the main Minitab folder

  If the macro is in a different location, you can specify the folder by including a path when you invoke the macro. For example `%c:\SALES\ANALYZE`.

  Note To specify the **Default file location** or the **Macro location**, choose **File** > **Options**, select **General**.
- If a file name includes spaces, put the name in single quotes, as in:
  `%'a very long file name.MAC'`

Adding Control Statements

Control statements can make your macro flexible and powerful. For example, use control statements for the following tasks:

- To perform some action only if some condition is true or false, use an `IF` statement.
- To perform some action a set number of times, use a `DO–ENDDO` loop.
- To repeat a block of commands as long as some condition is true, use a `WHILE–ENDWHILE` loop.
- To start another macro from within your macro, use `CALL` and `RETURN`.

More about Control Statements on page 25

Adding Comments and Notes

You can annotate your macro program by using the comment symbol `#` and the `NOTE` command.

Add comments that do not display in the output

Use comments (#) to make your macro file more readable with spaces or to add helpful notes to yourself.

- Simply place the symbol # anywhere on a line to tell Minitab to ignore the rest of the line.
- Text after # is not displayed in the output when the macro is executed (even when you use `ECHO`).

Tip You can also make your macro file more readable by adding blank lines between the lines of macro statements and commands. The blank lines will not interfere with the execution of the macro, and will not appear in the output. You do not have to start a blank line with a # symbol.

Add notes that display in the output

Use the `NOTE` command to make your output more readable with spaces or to add helpful notes to yourself. Text from notes will appear in separate output tabs unless you use the `MTITLE` and `ENDMTITLE` commands to specify a block of output. For more information, go to NOTE: Session command for adding comments that are displayed in the output on page 53.

Macros that Start Automatically

You can create a special file called STARTUP.MAC which executes automatically every time you start or restart Minitab. A startup macro is a handy tool if you wish to avoid typing the same commands every time you start a Minitab session.

STARTUP.MAC can be a global macro or local macro. Users of earlier versions of Minitab may have an Exec file named STARTUP.MTB which serves the same purpose and will still work.

To create a macro that starts automatically

Create your macro with session commands using a text editor or Minitab. The macro can be written as a global macro, a local macro, or an exec.

Save the macro file in text-only format, with the file name STARTUP. Use the file extension .MAC for a global macro or a local macro. Use the file extension .MTB for an exec. Save the file to the Macros subfolder of your main Minitab folder.

If you want, you can save your startup macro to a different folder. Minitab looks for macro files in the following order:

1. The folder where the project is.
2. The **Default file location** for Minitab
3. The **Macro location** for Minitab
4. The Macros subfolder of the main Minitab folder

Minitab executes the first file it finds. Files with the extension .MAC take precedence over files that begin with .MTB.

Note To specify the **Default file location** or the **Macro location**, choose **File** > **Options**, select **General**.

Finding Problems in Macros

If your macro produces unexpected results or generates an error message, Minitab provides several tools to help you track down and correct the problem. You should check for, and correct, these common problems first:

- The syntax used in the macro is not correct – for example, the macro does not begin with `GMACRO` or end in `ENDMACRO`.
- The Minitab commands in the macro are not correct – for example, a command is misspelled, or a column name is provided when the command expected a constant. This kind of mistake generates the same kind of error message you would have received if you were using Minitab in interactive mode.
- The macro uses a Minitab command that works differently in a macro than in interactive Minitab.

Advanced Macros

Advanced Macros

Local macros are more complex than global macros, and thus harder to write. However, they are more powerful and flexible. If you need to write a fairly complex macro, or if you want a macro that you can execute like a Minitab command, then you should write a local macro.

Local macros can use temporary variables, arguments, and subcommands to enhance the processing capabilities of the macro. Local macros also have a different structure that allows you to include areas for defining the common commands and the variables.

Local Macro Elements

Local macros have the capability to handle several elements which improve the processing capabilities of your macro. These following three elements are explained further:

Variables – Using Variables on page 15

Arguments – Using Arguments on page 16

Subcommands – Using Subcommands on page 17

Local Macro Structure

Template

Global macros use the template for naming purposes. Local macros use the template for naming the macro, but more importantly use the template for storing commands, subcommands and arguments. For more information, see Writing a Template on page 22

Declaration statements

The data variables that are used throughout a local macro need to be defined as columns, constants, or matrices. Declaration statements define the variable data type. For more information, see Declaration Statements on page 49

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, or a text string that is the name of the test.

Variables can be utilized in a local macro argument to allow you to enter data as the macro is invoked. They can also be used in a local macro control statement (found in the body of the macro) to enable complex calculations and data manipulations. All types of variables have to be declared in a declaration statement.

Variables for arguments

With global macros, you must provide the specific location, or specific value, of the data that needs to be processed from the command each time a macro is created. The data can not be changed when the global macro is invoked. A local macro can use variables to establish data unknowns that are determined when the macro is invoked. These variables are determined in the macro template, and are considered arguments. For more information on templates, see Writing a Template on page 22. For more information on arguments, see Using Arguments on page 16.

Variables for control statements

Local macros also allow you to use temporary variables that are known only to the macro and that are stored in the local worksheet. These temporary variables exist only while the macro is running. They are defined and manipulated using control statements within the body of your macro.

The only way you can utilize results within interactive Minitab or in a global macro 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 on the local worksheet and manipulate them as you wish, without affecting your regular worksheet at all. When you exit the local macro, the local variables disappear. These temporary variables are especially useful for performing calculations and using control statements. For more information on control statements, see Control Statement Overview on page 25.

Declaring Variables

In order to use argument or control statement variables, you must first declare the data type of the variable. The data can be text, suffixed, or unknown (considered "free") for all variables. For more information on declaring variables see Declaration Statements on page 49.

Naming variables

You should choose a variable name that represents the value that is going to take the place of the variable when the macro is invoked. The following rules apply for naming variables:

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

Special variables

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

Variable Type	Declare with	Contents	For more information see
Subcommand	Do not declare	An *implicit constant* that has a value of either 1 (if the subcommand was invoked) or 0 (if the subcommand was not invoked)	Determining whether or not the subcommand invokes on page 18
Text	MCONSTANT	A text constant that contains a text string	Using Text Data on page 47
Suffixed	MCOLUMN MCONSTANT	A range of columns or constants	Using Suffixed Variables on page 23
Free	MFREE	Column, constant, or matrix whose type is undetermined until the macro is invoked	Using Free Variables on page 50

Using Arguments

Arguments are variables that are passed into and out of a macro when it is invoked. The variables 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. An argument can be a variable which represents:

- a stored column, constant, or matrix from a global worksheet: 'Sales', C1, K2, or M1
- a number such as 2.3

Suppose that you want a macro that will draw a scatter plot with a fitted regression line and 95% confidence bands. Using a global macro for this situation would require you to specify, or predetermine, which columns contain the data while creating the macro. While invoking the global macro, you would not be able to specify different columns for the command.

However, with a local macro, you could specify which columns to use either when you create the macro, or when you invoke the macro using variable arguments. The undetermined column specification variables, used when creating the local macro for this situation, are examples of arguments. They allow you to enter whatever columns you wish 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. In addition, arguments can tell the local macro where to store results when the macro is finished processing.

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 under the **Constants** heading of **Worksheet Information**. Right-click on the worksheet tab to see **Worksheet Information**.

Example of a macro template with arguments

The three arguments in the following template are `X`, `XBAR`, and `PCT`. `X` is a column that contains the data, `XBAR` is the constant where the answer will be stored, and `PCT` is an optional constant that affects the subcommand. All three arguments will be given specific values when the macro is invoked.

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

Using Subcommands

Local macros can also 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 local macro.

To add subcommands to a macro

- Write a template that includes a subcommand.
- If any of your subcommands include arguments, you must declare the variable data type for those arguments in the declaration statements.
- If any of your subcommands include arguments that are constants, you can assign default statements to those arguments in the body of the macro.

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, thereby accepting the default, or use it with two arguments. You cannot use the subcommand with the data value for one argument and take a default for the other argument.

Example of creating and invoking a macro with a subcommand

The following macro, `TRIM2`, includes 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`.

```
MACRO
TRIM2 X XBAR;
      PERCENT PCT.
    #
    # TRIM2 takes one column from the global worksheet, X, as input.
    #The column must already contain data. The macro 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
    DEFAULT     PCT = 5
    #
    # First we calculate the trimming points T1 and T2.
    MTITLE "Trimmed Mean"
    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;
      Exclude;
      Rows 1:T1 T2:N.
    ENDIF
    LET XBAR = MEAN(XTRIM)
    PRINT XBAR.
    ENDMTITLE
     ENDMACRO
```

Then 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. 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.

Here is what you would type for invoking the macro:

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

Determining whether or not the subcommand invokes on page 18

Assigning default values to subcommand arguments on page 40

Determining whether or not the subcommand invokes

As with regular Minitab commands, subcommands of macros are optional – when invoking the macro, you 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.

If you type the `PERCENT` subcommand while invoking the macro below, Minitab sets the variable subcommand constant equal to 1, thereby leaving the percent value up to you. If you do not type `PERCENT`, the variable subcommand constant defaults to 0, thereby accepting the percent value. 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
MCOLUMN    X XSORT XTRIM
DEFAULT     PCT = 5
```

```
body of the macro
IF PERCENT = 0
  NOTE Trimming 5 percent from each end
ENDIF
ENDMACRO
```

Local Macro Structure

Local macros are created in the same way as global macros, using a text editor or various features of Minitab. For more information, go to Global Macro Structure on page 11. However, the structure and the contents of a local macro can differ significantly.

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. The contents of a local macro follow this structure:

```
MACRO
template

declaration statements

body of the macro
ENDMACRO
```

MACRO and ENDMACRO

`MACRO` and `ENDMACRO` mark the beginning and end of each macro. You can have more than one macro within a local macro file – see Invoking Macros from within Macros on page 26. `MACRO` must be the first line of your macro because it labels the macro type as local, not global. `MACRO` and `ENDMACRO` can not be abbreviated.

Template

The template gives the macro command name and any subcommands, as well as any undetermined arguments. For more information, go to Writing a Template on page 22.

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. For more information, go to Declaration Statements on page 49.

Body of the macro

The body of a macro consists of command language that controls the automatic data processing. The language includes:

- Minitab commands
- Control statements
- Macro statements (such as `IF`, `THEN`, `PAUSE`, `CALL` and `GOTO`)
- Invocation of other global macros

Invoking a Local Macro

1. In the **Command Line** pane, enter the percentage symbol % followed by the macro file name, as in `%TRIM`. Also consider the following issues:
   - The default file name extension for local macros is .MAC. When you invoke a macro that has an extension of .MAC, you only need to type the file name, as in `%TRIM`. If the extension is not .MAC, you must type the file name and extension, as in %TRIM.TXT.
   - If you invoke a macro that is one of the following folders, then you do not have to specify the file path when you run the macro:
     - The folder where the project is
     - The **Default file location** for Minitab
     - The **Macro location** for Minitab
     - The Macros subfolder of the main Minitab folder

     If the macro is in a different location, you can specify the folder by including a path when you invoke the macro. For example `%c:\SALES\ANALYZE`.

     Note To specify the **Default file location** or the **Macro location**, choose **File** > **Options**, select **General**.
   - If a local macro file name includes spaces, put the name in single quotes, as in `%'a very long file name.MAC'`
2. After the file name, type any undetermined arguments which belong with 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 optional subcommands, type them as in interactive Minitab, ending each line with a semicolon or a period, as in
   ```
   %TRIM C1 K2;
   PERCENT 4.
   ```

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.
#
```

(3)

```
MCONSTANT N T1 T2 XBAR
MCOLUMN X XSORT XTRIM
```

(4)

```
#
# first we calculate the trimming points T1 and T2
MTITLE "Trimmed Mean"
LET N = COUNT(X)
LET T1 = ROUND(N*0.05)
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;
    EXCLUDE;
    ROWS 1:T1 T2:N.
ENDIF
LET XBAR = MEAN(XTRIM)
PRINT XBAR.
ENDMTITLE
```

(5)

```
ENDMACRO
```

Key

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 22.

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 49.

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 and Notes on page 13.

Writing a Template

A global macro template simply names the group of macro commands, whereas a local macro template lists the name and the macro command language. While the local macro template does not include macro statements or control statements, it does contain the command, its subcommands and any associated arguments.

Template Requirements

The first line of the template contains the macro name. You should use the same name for the template as the file name, unless you intend on using the template for multiple macro files. The file name is used when you invoke a macro, whereas the template name is used in constructing a macro file.

The only lines that can appear between the word `MACRO` and the template are comment lines that begin with #.

Command and subcommand arguments must have legal variable names. For more information, go to Using Variables on page 15.

Only the first four letters of macro subcommands are used by Minitab.

You may have two or more macros in one file. Each macro must follow the local macro structure, and each must have a unique template name. When you invoke the macro containing multiple macros, Minitab executes the first macro in the file. You can invoke subsequent macros within the file by using a CALL statement with each template name.

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	Invoked by
`Trim X Xbar`	`%TRIM C5 K1`

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.

Example of a template for a command with a subcommand

Template	Invoked by

`Trim X Xbar;`
`  Percent Pct.`
	`%TRIM C1 C5;`
	`  PERCENT 5.`

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.

Declaration statements

All variables used in a local macro must be declared. Declaring a variable tells the local macro what type of variable to expect from the user, or the macro, while invoking. For information on the commands that you use to declare variables, go to MCONSTANT, MCOLUMN, MMATRIX, and MTYPE: Session commands for declaring variables on page 49.

Using text

You can use text data in columns, in stored constants, and as text strings in all three types of macros. In addition, you can pass a text string into a macro by enclosing the string in double quotes when invoking the macro. The passed string can then 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 could be created in a local macro. For more information on commands that work with text, go to KKCAT, KKNAME, and KKSET: Session commands for using text on page 47.

Using free variables

You may want a local macro to operate with a column, constant, or matrix—whatever the user decides to use when he or she invokes the macro. The local macro can then take appropriate action, depending on the type of variable used when invoking the macro. A free variable is an argument variable whose type—column, constant, or matrix—is not determined until the macro is invoked. For information on how to use a free variable, go to MFREE: Session command for declaring a free variable on page 50.

When you have a free variable, use the `DTYPE` command to determine the type of variable. For more information, go to DTYPE: Session command for determining the data type of a column or a constant on page 41.

Using Suffixed Variables

A suffixed variable is a variable that represents a range of values. The range can include columns and constants. Suffixed variables are most useful in the following cases:

- 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, depending on what data is applicable.

Suffixed Variable 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.

Variable Name	**Period**	**Suffix**	**Suffixed Variable**	**Range of Suffixed Variables**
X	.	1	X.1	X.1-X.5
My_Data	.	1	My_Data.1	My_Data.1-My_Data.5
Test	.	1	Test.1	Test.1-Test.testnum
Test	.	testnum	Test.testnum	

The variable name and the suffix can each have up to eight characters. However, only the last eight characters of a suffixed variable, including the period, are shown when a suffixed variable is printed. So if you plan to print out suffixed variables, you should probably keep them short, as in Col.1-Col.5 or X.1-X.N.

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 variable type.

	Templates (where `TRIM` is the command name)	Declarations
Legal:	`TRIM X.1-X.5`	`MCOLUMN X.1-X.5`
	`TRIM X.1-X.5 Y.1-Y.8`	`MCONSTANT X.1-X.5 Y.1-Y.8`
	`TRIM Z X.3-X.20 W1 W2`	`MCOLUMN Z X.3-X.20 W1 W2`
Illegal:	`TRIM X.1-X.3 X.4-X.5`	`MCOLUMN X.1-X.3 X.4-X.5`
	`TRIM X.1-X.2 Y X.3-X.5`	`MCONSTANT X.1-X.2 Y X.3-X.5`
	`TRIM X.5-X.1`	`MCOLUMN X.5-X.1`

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

Legal template statements:	`MYPROG1 X.1-X.10 Y.1-Y.N`
	`MYPROG2 X.1-X.10 Y.4-Y.20`

	`MYPROG3 X.1-X.M;`
	`  SUB1 Y.1-Y.N;`
	`    SUB2 Z.5-Z.P W.1-W.10.`

Illegal template statement:	`MYPROG4 X.1-X.M Y.1-Y.N`

Once you have declared a suffixed variable, you cannot declare another variable with the same prefix, even one of the same type. The following two declarations cannot be used in the same program:

```
MCOLUMN X.1-X.N
MMATRIX X
```

Because the prefix "X" is used with `MCOLUMN`, it cannot be used again – either for additional columns or for any other type of variable.

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 applies a value to N automatically 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. 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.

```
MACRO
GENMEDIANS MEDIANS
#
MCOLUMN X.1-X.5 MEDIANS
#
RANDOM 100 X.1-X.5
RMEDIAN X.1-X.5 MEDIANS
ENDMACRO
```

Suppose you stored this macro in a file called GENMEDIANS.MAC, and invoke it with `%GENMEDIANS 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
RMEDIAN 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, medians appear in 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. The macro requires data in the global worksheet so that you can specify columns for X.1-X.N.

```
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
#
NAME MIN "Min"
NAME MED "Med"
NAME MAX "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-C13, and store them in C21, C22, and C23. When invoking the macro, we would type `%ORSTATS C5-C13 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, ... , C13 to X.8 and C21 to `MIN`, C22 to `MED`, and C23 to `MAX`.

Controlling Macro Flow

Control Statement Overview

Control statements can make your macro more flexible and powerful because they allow you to control the sequence in which commands in the macro are executed. They can perform some action given a condition using an `IF` statement. They can perform some action repeatedly using a `DO-ENDDO` loop statement. They can start other macros from within a macro using a `CALL` and `RETURN` statement. The following pages document these control statements, and more.

You can also nest control statements. For example, one control statement, such as an IF statement, can contain several other control statements, such as additional IF statements or a DO statement.

Commands

IF, ELSEIF, ELSE, ENDIF: Session commands for executing code depending on a logical condition on page 46

DO and ENDDO: Session commands for looping through a block of commands on page 40

WHILE and ENDWHILE: Session commands for repeating a block of commands depending on a logical expression on page 55

NEXT: Session command for transferring control from a loop to the beginning of the block on page 52

BREAK: Session command for transferring control from a DO- or WHILE-loop on page 36

GOTO and MLABEL: Session commands for branching to any line in a macro on page 44

CALL and RETURN: Session commands for passing control to another macro on page 38

EXIT: Session command for transferring control back to Minitab or for closing Minitab on page 42

CD: Session command for displaying or changing the current directory on page 39

Invoking Macros from within Macros

You may have two or more macros in one file. Each macro in the file follows the usual structure (beginning with GMACRO or MACRO, ending with ENDMACRO, etc.), 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. There are some restrictions on which type of macro another macro can call:

From within this type of macro	You can invoke...		
Global	Global		Exec
Local		Local	
Exec	Global	Local	Exec

You invoke a macro from within a macro in the same way you invoke a macro from the Command Line pane. On a line, put the symbol % followed by the name of the macro file, as in %TRIM. You can also include a path statement, as in %C:\MYWORK\TRIM. If it is a local macro, include all appropriate arguments and subcommands.

Because the macros you execute are stored in your worksheet area, the only limitation to the number of macros you can nest is the amount of space available in your worksheet.

The following example removes data from the analysis when a data set is too small to analyze. Three macros are stored in separate files. The main file, stored as ANALYZE2.MAC, determines how many observations are in the data set. If there are fewer than 5, it invokes the macro file TOOSMALL.MAC. TOOSMALL prints out a message then prints the data set. If the data set has at least 5 observations, ANALYZE2 invokes the macro file OK.MAC. OK is similar to ANALYZE in Creating a Global Macro on page 11.

ANALYZE2.MAC

```
GMACRO
ANALYZE2
#
LET K90 = COUNT(C1)
IF K90 < 5
  CALL TOOSMALL
ELSE
```

```
  CALL OK
  ENDIF
ENDMACRO
```

TOOSMALL.MAC

```
GMACRO
TOOSMALL
#
MTITLE "Not Enough Data ";
  NOTITLE.
PRINT "Data set has fewer than 5 observations."
PRINT "No analysis will be done. Your data is listed below."
PRINT C1 - C3
ENDMTITLE
ENDMACRO
```

OK.MAC

```
GMACRO
OK
#
NAME C1 'Yield' C2 'Chem1' C3 'Chem2' C5 'Ln.Yield'
MTITLE "Your data is listed below ";
  NOTITLE.
PRINT C1-C3
ENDMTITLE
DESCRIBE C1-C3
LET C5 = LOGE('Yield')
REGRESS;
 RESPONSE C5;
 CONTINUOUS C2 C3;
 TERMS C2 C3.
ENDMACRO
```

Managing Input and Output

Data management overview

You can pass information through a macro using arguments, or you can pass information through macros by providing user interaction. Arguments can only be used in local macros and they are often not very user friendly. Instead, you can provide questions or messages that interact with the user of the macro. Minitab provides several communication aids that are compatible with global macros and that provide user friendliness: the command `NOTE`, a special "`TERMINAL`" option on `WRITE`, `READ`, and `SET`, and the statement `PAUSE`.

You can also manipulate the macro output using several Minitab commands. You can suppress your output using `BRIEF`. You can control graph output using commands such as `NOFRAME`, `GSAVE`, `GSCALE`, or `NOBRUSH`. You can also change or add an argument name or title.

Commands

NOTE: Session command for adding comments that are displayed in the output on page 53

READ, TSET, and SET: Session command for asking users questions and using the answers in a macro on page 54

PAUSE and RESUME: Session commands for pausing and resuming a macro on page 53

BRIEF: Session command for controlling the amount of output on page 36

MTITLE: Session command for adding a title above output on page 52

WTITLE: Session subcommand for specifying the title of the output pane on page 56

GSAVE: Session subcommand for saving a graph in a file on page 44

GSCALE: Session command to determine appropriate scaling for a graph on page 45

Prompting a user for information

`READ` and `SET` have a special feature that allows you to ask users questions and then use their answers in the macro. A macro will pause for user input if you use `READ` or `SET` with the subcommand `FILE` with the special file name TERMINAL. TERMINAL tells Minitab to wait for input from the keyboard. `READ` and `SET` also have other subcommands.

Handling Macro Errors

Handling Errors Overview

INFO: Session command for summarizing the current worksheet on page 47

ECHO and NOECHO: Commands for displaying Minitab commands from macros and execs on page 42

DEBUG and NODEBUG: Session commands for finding problems in macros on page 39

PAUSE and RESUME: Session commands for pausing and resuming a macro on page 53

Interpreting Error Messages

Minitab has an internal program called a macro processor that handles all the work that is specific to macros. The macro processor monitors which macro file you are currently using and what macros are in the file, and it processes all macro statements.

Error messages can be sent from the macro processor to the Minitab program. When the macro processor encounters a Minitab command, the processor checks the command briefly and then gives the command to the Minitab program to fully check and execute. Knowing where a message came from can help you troubleshoot:

    ** ERROR (two asterisks) means an error was found by the macro processor

    * ERROR (one asterisk) means an error was found by regular Minitab

Debugging Tools

"Debugging" is the art of finding problems (*bugs*) in a computer program. You can use several techniques and commands to display information about the macro such as `ECHO` and `DEBUG`. You can also pause the macro so you can investigate problem areas using `PAUSE` and `RESUME`.

Commands that Work Differently in Macros

One source of errors can be Minitab commands that work differently in macros than they do in interactive Minitab.

Commands that work differently in global and local macros

- `READ` and `SET`:
  - If your macro includes data after these commands, you must use the command `END` on the next line following the data.
  - If you use the `FORMAT` subcommand with these commands, the `END` command must be at the beginning of the next line following the data. If you indent the `END` command at all, Minitab will not recognize it and you will get an error message.
  - If you use `READ` or `SET` to input data from a file, you must specify the file name on the `FILE` subcommand. You cannot specify the file name on the main command as you can in interactive Minitab.
  - **In local macros:** If you see the error "Missing END for READ, or SET" it may be because you have named a local variable with the same name as a Minitab command, and entered it after `READ` or `SET`.

`BRIEF` and `ECHO` are commands that change output settings for the current macro.

Commands that work differently in local macros

- `LET`. You cannot use a Minitab function or column statistic as a variable name in a `LET` command. Thus
  ```
  LET Mean = X1 + X2 + X3
  ```
  is illegal because there is a Minitab function called `MEAN`. In general, it is better not to use Minitab command names as variable names in a macro.
- `ERASE`. Erases local worksheet variables, but it does not erase the declaration of a variable. That is, you cannot declare the same variable twice in one macro.
- `EXECUTE`. You cannot invoke `EXECUTE` from within a local macro. You can, however, invoke a local macro from within an Exec macro.
- `INFO`. In a local macro, `INFO` displays information on the local worksheet. For more information, go to Getting Information About the Local Worksheet on page 47.
- `SAVE` and `RETRIEVE`. You cannot use either of these commands in a local macro. To save data in the local worksheet, use the command `WRITE`.

Commands and subcommands that are not allowed in macros

Commands and subcommands that open projects or restart Minitab are not allowed in macros. Also, local macros cannot include commands that do the following:

- Make different worksheets active.
- Specify storage locations that are after the last column in use.

The following sections list some specific commands and subcommands that are either deprecated or not allowed in macros. Where subcommands are listed, the command itself is permitted. For example, you can use the COPY command in local macros, but you cannot use the NEWWS, AFTER, or STORE subcommands of COPY.

Not allowed in local macros

COPY: Allowed

  NEWWS: Not allowed

  AFTER: Not allowed

  STORE: Not allowed

EXECUTE: Not allowed

NEW: Not allowed

RETRIEVE: Not allowed

SAVE: Not allowed

SORT: Allowed
- NEWWS: Not allowed
- AFTER: Not allowed

SPLIT: Not allowed

STACK: Allowed
- NEWWS: Not allowed

SUBSET: Not allowed

TRANSPOSE: Allowed
- NEWWS: Not allowed
- AFTER: Not allowed

UNSTACK: Allowed
- NEWWS: Not allowed
- AFTER: Not allowed

WOPEN: Not allowed

WORKSHEET: Not allowed
- RENAME: Allowed

Not allowed in global macros

NEW: Allowed
- PROJECT: Not allowed

RESTART: Not allowed

Using Execs

Execs overview

Execs are stored commands that you use over and over, so that you do not have to retype the commands each time. You can even write an interactive Exec, which pauses during execution, prompts the user for information, then continues with execution. Execs are useful for many things, including the following:

- Repeating a block of commands many times, which is useful for simulations
- Looping through columns of the worksheet, doing the same analysis on each block of columns
- Looping through rows of the worksheet, doing the same analysis on each block of rows
- Performing complex operations not provided as stand-alone commands

How Execs are different from global and local macros

Global and local macros are more powerful and flexible than Execs. Other differences include the file extension and how you invoke the macro:

- The default extension for an Exec is .MTB. The default extension for global and local macros is .MAC.
- You invoke an Exec by running the command `EXECUTE` or by choosing **File** > **Run an Exec**. You invoke a global or local macro by entering the symbol % followed by the macro file name. For example, `%SALES` invokes the macro SALES.MAC.

If you have Execs that were written using previous releases of Minitab, you can continue to use them with no change, unless, of course, the Execs use deprecated commands.

Converting Execs to global or local macros

To convert your Exec to a global macro

1. Add three lines to your Exec file: `GMACRO` as the first line, `ENDMACRO` as the last line, and the template (the macro name) as the second line of the file.
2. Check for Minitab commands that work differently in global macros (below).
3. Save the macro as a text file, with the extension .MAC.

Note For global macros and local macros, the decimal separator is always a period (.) and the list separator is always a comma (,). Change these symbols if necessary.

Once you have converted your Exec to a global macro, you can incorporate any of the features documented in the chapters for global macros such as `DO`-loops and `IF` statements. You can also include several global macros within one global macro file.

Converting your Exec to a local macro

Local macros do not support the `CK` capability, which is a specialized looping feature exclusive to Execs. If your Exec uses the `CK` syntax, replace the syntax with the appropriate control statement.

Commands that work differently in global and local macros

- Execs allow a repeat factor, such as "3" in the command `EXECUTE "MYMACRO" 3`. Global macros do not allow a repeat factor because they allow control statements such as `DO`-loops and `WHILE` statements which work much more efficiently. If your Exec requires such a repeat factor, you will need to incorporate that operation within the body of the global macro.
- In earlier releases of Minitab, the default was `ECHO`. Now the default is `NOECHO`, which means that commands are not normally displayed while the macro executes. If your Exec contains `NOECHO` commands, there is no harm in leaving them there, but they may not be necessary anymore.
- `READ` and `SET` commands should follow these conventions:
  - If the command reads data from a file, you must modify the command so that the file name is listed with a `FILE` subcommand, rather than being listed on the main command line.
  - If the command is followed by data, you must include the statement `END` at the end of the data, on its own line.
  - If the command is followed by a `FORMAT` subcommand followed by data, the `END` statement must begin at the beginning of the line. If `END` is indented at all, Minitab will not recognize it and you will get an error message.

For a longer list of commands that work differently in global and local macros, go to Commands that Work Differently in Macros on page 28 and Commands and subcommands that are not allowed in macros on page 29.

Creating an Exec

With a text editor, such as Notepad, store the file in a text format. Save the file with the extension .MTB; that way, when you use the `EXECUTE` command, you will not have to type the extension because Minitab will assume the file has the default extension of .MTB.

Example of Exec

Each month, a laboratory sends you data on three chemical measurements: Yield, Chem1, and Chem2. You always do the same analysis: descriptive statistics, plots of Yield versus the two other measures, a regression, and a residual plot. Suppose you use your computer's editor to create the following file called ANALYSIS.MTB:

```
NAME C1 'Yield' C2 'Chem1' C3 'Chem2'
DESCRIBE C1-C3
LET C5 = LOGE('Yield')
PLOT C1*C2
PLOT C1*C3
REGRESS;
 RESPONSE C5;
 CONTINUOUS C2 C3;
 TERMS C2 C3;
 RESIDUALS C10;
 FITS C11.
NAME C10 'Resids' C11 'Fits'
PLOT C10*C11
```

Then, if you put the data for January in the file JAN.MTW, you can perform your analysis by doing the following:

1. Choose **File** > **Open** and select JAN.MTW.
2. Choose **File** > **Run an Exec**. Click **Select File**.
3. Select ANALYSIS.MTB. Click **Open**.

Running an Exec

```
EXECUTE ["filename"] [K times]
```

This command runs commands that have been stored in a file. These files are Execs.

The default file extension for Execs is .MTB. When using `EXECUTE`, you do not need to type the file extension if it is .MTB. The default file name is Minitab.MTB – if you do not specify a file name with `EXECUTE`, Minitab looks for the file Minitab.MTB and runs the file if it exists.

The optional argument `K` lets you specify how many times to run the Exec. `K` can be any integer ≥ 1. The default value is 1, which means that the macro will be executed one time. If `K` > 1, the macro is executed `K` times.

To interrupt the execution of an Exec, press **Ctrl+Break**. Minitab will finish executing the command in process before it stops the macro.

Creating Loops

Looping through commands

Suppose you want to train your eye to judge normal probability plots. So you decide to generate 20 plots for data from a normal distribution. First store the following commands in a file called NPLOT.MTB:

```
RANDOM 50 C1
LET C2 = NSCORES(C1)
```

```
NAME C1 'Data' C2 'Nscores'
PLOT C1*C2
```

To execute this file 20 times, to get 20 different normal probability plots, type

```
EXECUTE "NPLOT" 20
```

You can also loop through rows of data. Suppose we have a full year of the laboratory data from our first example, one month stacked on top of another, in a file called LAB.DAT. There are now four variables, Yield, Chem1, Chem2, and Month. To do the same analysis as before, separately for each month, we store the following commands in the file YEAR.MTB:

```
NAME C11 'Yield' C12 'Chem1' C13 'Chem2' C20 'Resids' C21 'Fits'
COPY C1-C3 C11-C13;
  INCLUDE;
    WHERE "C4 = K1".
PRINT K1
DESCRIBE C11-C13
PLOT C11*C13
PLOT C11*C13
REGRESS;
  RESPONSE C11;
  CONTINUOUS C12 C13;
  TERMS C12 C13;
  RESIDUALS C20;
  FITS C21.
PLOT C20*C21
ADD K1 1 K1
```

Then, to analyze the file LAB, we type:

```
LET K1 = 1
READ C1-C4;
FILE 'LAB'.
EXECUTE "YEAR" 13
```

Looping through columns and matrices

A special feature, sometimes called the `CK` capability, allows you to loop through columns of the worksheet. Suppose you have a file, MYDATA.DAT, containing 21 variables and you want to plot the last variable versus each of the first twenty variables. That's twenty separate plots. First store the following commands in a file called PLOTS.MTB:

```
PLOT C21*CK1
ADD K1 1 K1
```

Then type:

```
READ C1-C21;
FILE 'MYDATA'.
LET K1 = 1
EXECUTE "PLOTS" 20
```

The first time through the loop, K1 = 1. This value is substituted for the `K1` in the `PLOT` command, giving `PLOT C21*C1`. The next time through the loop, K1 = 2, giving `PLOT C21*C2`, and so on.

Matrices also have this capability, using MK1. Stored constants do not.

The next example shows how to accumulate column statistics in one column. Suppose you have data in C1 through C30 and you want to compute the mean of each column and store those means in C40. Store the following commands in the file MEAN.MTB:

```
LET C40(K1) = MEAN (CK1)
ADD K1 1 K1
```

Then type:

```
LET K1 = 1
EXECUTE "MEAN" 30
```

The first time through the loop K1 = 1, so row 1 of C40 will equal the mean of C1. The next time through the loop K1 = 2, so row 2 of C40 will equal the mean of C2, and so on.

Using Conditional Execution

If the argument `K` on `EXECUTE` is zero or negative, the Exec is not executed. This feature allows you to do conditional execution. As an example, we will modify the Exec MEAN.MTB so that it accumulates means for just those columns that have more than 9 observations. We need two files. MEAN10.MTB contains:

```
LET K3 = (COUNT(CK1) > 9)
EXECUTE "OVER9" K3
ADD K1 1 K1
```

and OVER9.MTB contains:

```
LET C40(K2) = MEAN(CK1)
ADD K2 1 K2
```

To use this macro, we type:

```
LET K1 = 1
LET K2 = 1
EXECUTE "MEAN10" 30
```

First, notice that we have nested two Execs, that is, `MEAN10` calls (or executes) `OVER9`. Nesting helps you write fairly sophisticated Execs. You can nest up to five deep on most computers.

To see how this macro works, we will look at the first three columns. Suppose C1 has 23 observations, C2 has 7, and C3 has 35. When we first execute `MEAN10`, K1 = K2 = 1. Then K3 = 1 since `COUNT (C1)` > 9. Since K3 = 1, `OVER9` is executed once, `MEAN (C1)` is stored in row 1 of C40, and K2 = 2.

For the second time through the loop, K2 = 2 and K1 = 2. This time K3 = 0 since `COUNT (C2)` < 9, and `OVER9` is not executed. For the third time through the loop, K1 = 3 and K2 = 2. Then K3 = 1 since `COUNT (C3)` > 9, `OVER9` is executed, and `MEAN(C3)` is stored in row 2 of C40.

Handling Arguments

Sometimes you do not know how many columns of data will be used in each analysis; one time you may need the exec to operate on 10 columns, and the next time on 13 columns. The `CK` capability also allows you to write an exec that can operate on a variable number of columns.

For example, suppose each month a researcher collects data from tomato plants. Some months she has 20 plants, other months just 5. The data for one month consist of one variable for each plant. First she creates the following Exec, called PLANTS.MTB:

```
HISTOGRAM C1-CK50
DESCRIBE C1-CK50
ADD K50 50 K51
COPY C1-CK50 C51-CK51
 (etc.)
```

Then, if she has data on 13 plants, she types:

```
READ C1-C13
  (data)
END
LET K50 = 13
EXECUTE "PLANTS"
```

Interactive Execs

It is possible to write an Exec which will execute, pause for user input, and then continue executing. This is accomplished by using the special file name TERMINAL with the `READ` and `SET` commands.

Here is an example. We have two Execs. The first, PLANTS.MTB, is the same as described in Handling Arguments on page 34. The second, TOMATO.MTB, contains:

```
NOTE How many tomato plants do you have this month?
SET C50;
  FILE "TERMINAL";
  NOBS 1.
COPY C50 K50.
EXEC "PLANTS"
```

When you type `EXECUTE "TOMATO"`, the note "How many tomato plants do you have this month?" is printed. The dialog waits for you to respond. You type a number and press **Enter**. The subcommand `NOBS = 1` tells `SET` to expect just one number. This means the user of the macro does not have to type the word `END` to signal the end of typing data to `SET`. The macro `TOMATO` is then executed with the correct number of plants. The command `NOECHO` suppresses the echo printing of commands, and `ECHO` turns it back on.

Alphabetical list of macro commands

BREAK: Session command for transferring control from a DO- or WHILE-loop

Transfers control from within a DO- or WHILE-loop to the command immediately following the end of the loop. Thus BREAK breaks out of the loop.

The following is a simple example of BREAK in a global macro. The program works on a worksheet where one of the columns has the name X. The program goes through the values of X until it finds a missing value. It then leaves the loop and goes to the statement following ENDDO—in this example, DELETE. Note that this program does not handle the case when X has no missing values correctly. For an example that handles the case when X has no missing values, go to EXIT: Session command for transferring control back to Minitab or for closing Minitab on page 42.

```
GMACRO
NOMISS
#
# Takes data from the column named X. Finds the first missing
# observation. Then deletes all observations starting with the
# first missing to the end of the column.
# Constants K90 and K91 are used for scratch work
#
LET K90 = COUNT('X')
DO K91 = 1:K90
  IF  'X'(K91) = '*'
    BREAK
  ENDIF
ENDDO
DELETE  K91:K90 'X'
ENDMACRO
```

BRIEF: Session command for controlling the amount of output

BRIEF K

Controls the amount of output. For example, the following table describes how BRIEF works with most commands that create a designed experiment.

Value of K	Output that is displayed
0	Minitab displays no output from the command, but performs all specified storage and displays the following output: error messages, warnings, prompts, and notes; graphs; WRITE to the screen.
1	Minitab displays a summary of the design.
2 (default)	Same as K = 1 output.
3	Same as K = 2 output, but Minitab also displays the design table.

Used as a main command, BRIEF affects the amount of output produced by subsequent commands. Used as a subcommand, BRIEF only affects output for the command it is used with.

Most commands are affected by BRIEF only when it is set to 0. However, BRIEF affects the amount of output produced by the following commands in specific ways. Run the command HELP in the **Command Line** pane to open a PDF file with information on specific commands.

ARIMA

BBDESIGN

BLOGISTIC

CCDESIGN

CLUOBS

CLUVARS

DISCRIMINANT

EVDESIGN

FACTOR

FFDESIGN

GLM

KMEANS

LREGRESSION

LTABLE

LTEST

MIXREG

NLOGISTIC

OLOGISTIC

OPTDES

PROBIT

RLINE

RSREG

SCDESIGN

SLDESIGN

CALL and RETURN: Session commands for passing control to another macro

CALL *template*

RETURN

You can include several macros in one file, just as a program often includes several subroutines. CALL and RETURN let you specify when to pass control to another macro and when to return to the main macro. You can include several global macros in one file, or several local macros in one file, but you cannot mix global and local macros together in one file.

When you invoke a macro, from interactive Minitab or from another macro, the first macro in the file is executed first. Use the macro statements CALL and RETURN to invoke a different macro within the macro file.

Recall that the second line of a macro is the template, or the macro name. When one macro in a macro file calls another macro in that file, use the command CALL, followed by the name on that macro's template. If it is a local macro, include appropriate arguments and subcommands. Any macro in a macro file can CALL any other macro in the file, any number of times.

RETURN says to leave the current macro and go back to the calling macro, to the statement just after the CALL. RETURN is optional. If RETURN is not present in the macro that was called (the subroutine), then, after it has executed, control is transferred back to the calling macro.

The following example is a variation on ANALYZE2.MAC. This example, ANALYZE3, uses the TSET command to ask the user whether to print all the data. If the response is "yes", then the macro sets K80 to 1. If the answer is anything else, then the macro sets K80 to 0. For more information, go to READ, TSET, and SET: Session command for asking users questions and using the answers in a macro on page 54. The OK subroutine checks the value of K80 with an IF statement. If K80 equals 1, then the RETURN statement sends control back to the main macro. If K80 is anything else, then the macro prints one more note. When the ENDMACRO statement is encountered in either the TOOSMALL or OK subroutine, control is transferred back to the calling macro.

```
GMACRO
ANALYZE3
#
MTITLE
NOTE Do you want all the data printed?
NOTE Type "yes" or "no" in quotation marks,
NOTE then click the Submit button.
tset c10;
file "terminal";
end.
IF c10(1) = "yes"
 LET k80 = 1
ELSE
 LET k80 = 0
ENDIF
ERASE c10.
# If user types "yes" K80 = 1, if "no" K80 = 0
LET K90 = COUNT(C1)
IF K90 < 5
  CALL TOOSMALL
ELSE
  CALL OK
ENDIF
#
IF K80 = 1
NOTE Here are the data.
PRINT C1-C3
ENDIF
ENDMTITLE
ENDMACRO
```

```
#
#
GMACRO
TOOSMALL
MTITLE
NOTE Data set has fewer than 5 observations.
NOTE No analysis will be done.
ENDMTITLE
ENDMACRO
#
#
GMACRO
OK
MTITLE
NAME C1 'Yield' C2 'Chem1'  C3 'Chem2' C5 'Ln.Yield'
DESCRIBE C1-C3
LET C5 = LOGE('Yield')
REGRESS;
RESP c5;
CONT c2 c3;
TERMS c2 c3.
IF K80 = 1
  RETURN
ENDIF
NOTE Analysis done, but no data printed by request
ENDMTITLE
ENDMACRO
```

CD: Session command for displaying or changing the current directory

CD *[filepath]*

CD without a path displays the current directory. CD with a path changes the current directory to the one that you specify.

For example, `CD` displays the current directory, and `CD WILLIAMS\SALES91` changes the current directory to WILLIAMS\SALES91.

DEBUG and NODEBUG: Session commands for finding problems in macros

DEBUG

Displays information about the macro in the output.

NODEBUG (default)

Suppresses the display of information about the macro in the output.

DEFAULT: Session command for assigning default values to subcommand arguments

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.

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.

DO and ENDDO: Session commands for looping through a block of commands

DO K

ENDDO

Allows you to loop through a block of commands. K is set equal to the first number in the list, then the block of commands is executed. When Minitab reaches the ENDDO, K is set equal to the next number in the list and the block is executed again. This continues until all numbers in the list are used, or until you branch out of the DO-loop with a BREAK, GOTO, RETURN, or EXIT command.

The list of numbers can be an explicit list of any numbers or stored constants. A patterned list can be abbreviated using a colon and slash as in SET. For example, 1:10 is the list 1, 2, 3, ... , 10, and 1:1.8 /.2 is the list 1, 1.2, 1.4, 1.6, 1.8. Numbers can be in either increasing order or decreasing order. The following DO-loop changes the values in rows 1 through 10 and row 50 of columns C1 and C2 to the missing value code:

```
DO  K1 = 1:10 50
  LET C1(K1) = '*'
  LET C2(K1) = '*'
ENDDO
```

The following is a local macro that calculates a moving average of length three. It shows how to loop through the values in a column. Enter data in a column of the worksheet before you run the macro so that you can specify X.

```
MACRO
MOVAVE  X  Y
#
# Calculates the simple moving average of the data in X and
# stores the answer in Y.
#
MCONSTANT N  I
MCOLUMN   X  Y
LET N = COUNT(X)
LET Y(1) = '*'
LET Y(2) = '*'
DO I = 3 : N
  LET Y(I) = (X(I) + X(I-1) + X(I-2))/3
```

```
ENDDO
ENDMACRO
```

Note Instead of modifying a worksheet variable inside a DO/ENDDO loop, copying the worksheet variable to a local macro variable, modifying the macro variable in the loop, then copying the macro variable back to the worksheet variable might be faster.

DTYPE: Session command for determining the data type of a column or a constant

DTYPE *E* K

Use DTYPE to determine the data type of a column or constant (E), and store the results in a constant (K).

Value returned by DTYPE	Data type
0	Text
1	Real numbers
2	Integers
3	Date/time
10	Empty

DTYPE is often used with free variables (and the MFREE and MTYPE commands) in cases where the macro must be flexible enough to respond to a variety of possible inputs.

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.

Note DTYPE works only as a command. It does not work with IF or LET, for example.

Example of a macro that uses DTYPE

TELLDATA tells a user the data type of the variable that is specified when the macro is invoked.

```
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
#constants cannot be in date/time format
ELSEIF Vartype = 10
  NOTE Variable is empty
ENDIF
ENDMACRO
```

ECHO and NOECHO: Commands for displaying Minitab commands from macros and execs

The ECHO and NOECHO commands control whether commands in a macro or exec are in the output. When you develop a macro, you can use ECHO to see the commands so that you can find errors more easily.

You can submit ECHO and NOECHO before you invoke a macro. You can also place them anywhere within the body of a macro. You can use ECHO and NOECHO several times in a macro to turn on and off the display of commands.

`ECHO`

In ECHO mode, only commands in the body of the macro are in the output. The commands include Minitab commands, macro statements, and invocations of macros in other files. The commands do not include the template and declarations. (Declarations are in local macros only.) Text that is after a # is not in the output.

`NOECHO (default)`

In NOECHO mode, no Minitab commands or macro statements are in the output. The output of Minitab commands is in the output.

EXECUTE: Session command for running an Exec file

`EXECUTE` ***`"filename"`*** **`K`**

Note You cannot use EXECUTE in a local macro.

Runs commands that are stored in a file. These command files are called Execs.

You may specify the filename as either the name of the file in double quotes, or a stored text constant. The default file extension for Execs is MTB. When you use EXECUTE, you do not need to type the file extension if it is .MTB. The default file name is Minitab.MTB. If you do not specify a file name with EXECUTE, Minitab runs Minitab.MTB if it exists.

The optional argument `K` lets you specify how many times to run the Exec. `K` can be any integer. The default value is 1, which means that the Exec runs one time. If `K` > 1, the Exec runs `K` times. If K is < 1, the macro does not run. For information on using K to determine whether to run an Exec, go to Using Conditional Execution on page 34

To interrupt an Exec, press **Ctrl+Break**. Minitab will finish the command in process before it stops the macro.

EXIT: Session command for transferring control back to Minitab or for closing Minitab

`EXIT`

EXIT has two very different behaviors depending on whether it is used in global and local macros, or in an exec file, as follows:

- In a global or local macro, EXIT transfers control back to interactive Minitab.

- In an exec file, EXIT closes Minitab.

The following example is a modification of the macro NOMISS, which correctly handles the case when X contains no missing values. The program works on a worksheet where one of the columns has the name X. The program goes through the values of X until it finds a missing value. It then leaves the loop and goes to the statement following ENDDO—in this example, DELETE. If X has no missing values, the program prints a note.

```
GMACRO
NOMISS2
#
# Takes data from the column named X. Finds the first missing
# observation. Then deletes all observations starting with the
# first missing to the end of the column. Prints a message if
# the column has no missing values.
# Constants K90 and K91 are used for scratch work
#
LET K90 = COUNT('X')
DO K91 = 1:K90
  IF  'X'(K91) = '*'
    BREAK
  ENDIF
  IF K91 = K90
    NOTE Note: There are no missing observations in X.
    EXIT
  ENDIF
ENDDO
DELETE  K91:K90 'X'
ENDMACRO
```

GMACRO, MACRO, and ENDMACRO: Session commands for marking the beginning and ending of a macro

GMACRO

GMACRO must be the first line of your global macro. GMACRO specifies a global macro. GMACRO cannot be abbreviated.

MACRO

MACRO must be in the first line of your local macro, and specifies a local macro.

ENDMACRO

ENDMACRO ends all macros and must be in the last line of your macros. ENDMACRO cannot be abbreviated.

GOTO and MLABEL: Session commands for branching to any line in a macro

GOTO *number*

MLABEL *number*

Allows you to branch to any line in your macro. There can be several GOTO's in one program. A GOTO is matched to the MLABEL that has the same number. The number can be any integer from 1 to 8 digits long. It cannot be a variable.

The following example is a modification of the macro NOMISS, but uses GOTO instead of BREAK. The program works on a worksheet where one of the columns has the name X. The program goes through the values of X until it finds a missing value. It then leaves the loop and goes to the statement following ENDDO—in this example, DELETE. If X has no missing values, the program prints a note.

```
GMACRO
NOMISS3
#
# Takes data from the column named X. Finds the first missing
# observation. Then deletes all observations starting with the
# first missing to the end of the column.
# Constants K90 and K91 are used for scratch work
#
LET k90 = COUNT('X')
DO K91 = 1:k90
  IF  'X'(K91) = '*'
    GOTO 5
  ENDIF
  IF k91 = k90
  NOTE Note: there are no missing observations in X
  Exit
  ENDIF
ENDDO
MLABEL 5
DELETE  K91:k90 'X'
ENDMACRO
```

GSAVE: Session subcommand for saving a graph in a file

GSAVE *"file_name"*

GSAVE K

Saves the graph in a file.

The default file name is Minitab.PNG. You can specify a custom file name in double quotation marks ("file_name"), or as a stored text constant (K). You can also use any of the following subcommands to save the graph in a different graphics format.

Some graph commands—for example, HISTOGRAM C1 C2 C3—generate more than one graph. If you include the GSAVE subcommand with such a command, Minitab saves multiple files. Minitab gives each file a different file name. Minitab uses the first five characters of the name you specify, then appends a number (001, 002, and so on), for up to 300 files.

JPEG
JPEG color

PNGB
PNG grayscale

PNGC
PNG color

TIFB
TIF grayscale

TIF
TIF color

BMPB
BMP grayscale

BMPC
BMP color

GIF
GIF

EMF
EMF

RESOLUTION K
Saves the graph at a resolution of K dots per inch.

GSCALE: Session command to determine appropriate scaling for a graph

GSCALE K K

GSCALE is useful primarily when you are writing a macro that produces graphs, and you need to know information before you produce the graphs to ensure that the scaling on the graphs will look right. For example, you might want to generate two or more graphs that use the same scale, but you want to control the scale. You can use the data stored by GSCALE to specify scaling options in subsequent graph commands.

The arguments on the main command are the minimum (the first K) and the maximum (the second K) of the data from the columns to be graphed, combined. An easy way to get those values is to STACK all of the columns on top of each other in a new column, then use the MIN and MAX commands to store the minimum and maximum values.

NMINIMUM K
Specifies the minimum number of ticks to use.

NMAXIMUM K
Specifies the maximum number of ticks to use.

NTICKS K
Stores the number of ticks.

TMINIMUM K
Stores the minimum tick value.

TMAXIMUM K
Stores the maximum tick value.

TINCREMENT K
Stores the distance between ticks.

SMINIMUM K
Stores the scale minimum.

SMAXIMUM K
Stores the scale maximum.

IF, ELSEIF, ELSE, ENDIF: Session commands for executing code depending on a logical condition

```
IF logical expression

ELSEIF logical expression

ELSE

ENDIF
```

Allows you to execute different blocks of code depending on a logical condition. A logical expression is any expression from the LET command. The comparison and Boolean operators listed below are the features of LET that are most often used in IF.

Comparison and Boolean operators	Alternative form	Description
=	EQ	Equal to
~=	NE	Not equal to
<	LT	Less than
>	GT	Greater than
<=	LE	Less than or equal to
>=	GE	Greater than or equal to
&	AND	And
|	OR	Or
~	NOT	Not

In most cases the logical expression evaluates to a single number. If the number is 0 (false), the block of statements is skipped; if it is not 0 (true), the block is executed. If the logical expression evaluates to a column, then if all entries in the column are 0, the expression is considered false, otherwise it is considered true.

You can use multiple ELSEIF statements within the IF-ENDIF block.

The following is a simple example, using a global macro. Enter data in columns C1-C3 before you run the macro.

```
GMACRO
SMALL
#
# Takes the data in C1-C3. Finds the column with the smallest mean
# and prints that column. If, because of ties, there is no single column
# with the smallest mean, a message is printed.
#
LET K1 = MEAN(C1)
LET K2 = MEAN(C2)
LET K3 = MEAN(C3)
IF K1 < K2 AND K1 < K3
  PRINT C1
ELSEIF K2 < K1 AND K2 < K3
  PRINT C2
ELSEIF K3 < K1 AND K3 < K2
  PRINT C3
ELSE
  NOTE Note: There are ties.
ENDIF
ENDMACRO
```

INFO: Session command for summarizing the current worksheet

`INFO [C...C]`

Summarizes the current worksheet.

If no columns are specified, INFO prints a list of all columns used with their names and counts, all stored constants, all matrices. If there are missing observations, a count of these is also given. If a column contains text data, the letter T is printed to the left of the column. If columns have assigned formulas, these are printed along with the method selected for updating the calculations (manual or automatic). If you list columns, information is given on just those columns.

KKCAT, KKNAME, and KKSET: Session commands for using text

You can use text data in columns, in stored constants, and as text strings in all three types of macros. In addition, you can pass a text string into a macro by enclosing the string in double quotes when invoking the macro. The passed string can then 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 could be created in a local macro.

The following macro commands allow you to store text in a constant. They are especially useful for displaying titles and other annotation on macro output. The following text commands are used only in the body of global and local macros.

KKCAT K K K

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 following command KKCAT X Y Z would put the string "Mr.Jones" in constant Z.

KKNAME K C

Stores the name of column C in the constant K. For example, `KKNAME K1 C1` stores the name of column C1 in the constant K1.

KKSET K *"text"*

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. However, KKSET can store several text strings in several constants at once, whereas LET stores one text string in one constant. For example, `KKSET K1 "Text1" K2 "Text2"` stores the text strings in the constant K1 and K2.

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.

Example of a macro that uses text strings

The following local macro receives two strings when invoked and assigns them to constants.

```
MACRO
REVERSE file1 file2
#
# REVERSE reads the first 3 columns of the input file, file1.
#
MCONSTANT file1 file2
MCOLUMN X Y Z
PRINT file1 file2
#
# The FORMAT statement says that the data in file1 are text with 1 character.
# Valid rows would look like the following examples:
# a b c
# d e f
#
READ X Y Z;
  FORMAT(a1, x, a1, x, a1);
  FILE file1.
WRITE Z Y X;
  FILE file2.
#
# REVERSE now stores the 3 columns from file1 in reverse order as the output file,
file2.
#
ENDMACRO
```

Example of invoking a macro that uses text strings

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

```
%REVERSE "INPUT" "OUTPUT"
```

MCONSTANT, MCOLUMN, MMATRIX, and MTYPE: Session commands for declaring variables

All variables used in a local macro must be declared. Declaring a variable tells the local macro what type of variable to expect from the user, or the macro, while invoking.

Declaration requirements

- 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 plural synonyms MCONSTANTS, MCOLUMNS, and MMATRICES.) After the M- command, list all the variables that are of that type, separated by a space.
- An argument, which is a variable in the template, may be given the declaration MFREE. The variable data type—column, constant, or matrix—is determined by the type of the variable that is given when the macro is invoked. The macro statement MTYPE allows you to determine whether a variable declared with MFREE is a column, constant, or matrix.
- You may use a declaration statement several times, but only for different variables and only between the template and the body of the macro. Once a variable is declared, it cannot be redeclared. Variable declarations can only be made between the template and the body of the macro.
- The declaration commands (MCOLUMN, MCONSTANT, etc.) cannot be abbreviated.
- The declared variable must have a legal name. For more information, go to Using Variables on page 15.

Example of declaring variables

For example, suppose the template is as follows:

```
TRIM X Xbar
```

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 local macro file would have the following first few lines:

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

Note If you see an error that END does not follow READ or SET, a local variable could have the same name as a Minitab command. For example:

```
 SET col1
mini:maxi/1
END
```

where min and max are local variable names. Minitab interprets the second line as a command because MINI and MAXI 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 the variables in data entry.

Variable types

There are four special-purpose variables, which are each declared differently.

Subcommand
An implicit constant that has a value of either 1 (if the subcommand was invoked) or 0 (if the subcommand was not invoked).

Text
Declared with MCONSTANT as a text constant that contains a text string.

Suffixed
Declared with MCOLUMN or MCONSTANT as a range of columns or constants.

Free
Declared with MFREE as a column, constant, or matrix whose type is undetermined until the macro is invoked.

MFREE: Session command for declaring a free variable

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

Use a free variable in a macro

You must do five things in the local macro code 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 an additional variable as a constant: MCONSTANT Vartype
4. Use the macro statement MTYPE to analyze the free variable and store its variable type number in the constant declared in step. If the 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. You can include an MTYPE statement anywhere within the body of a local macro. For example, the command MTYPE X Vartype looks at the free variable X and stores its variable type (1, 2, or 3) in the constant Vartype.
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!, ENDIF.

Invoke macros that use free variables like any other 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 simple macro that uses free variables

The following local macro, TELLME, tells the user what kind of variable was used when the variable was invoked.

```
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!
ENDIF
ENDMACRO
```

You can invoke TELLME can be invoked in the following ways, which produce the following output:

Invoked like this	Produces this
%TELLME C1	X is a column!
%TELLME K1	X is a constant!
%TELLME M1	X is a matrix!

Example of a more complex macro that uses free variables

In the following local macro, BETWEEN.MAC, the arguments LOW and HI can be either columns or constants. Enter data in the worksheet before you run the macro so that you can specify X.1-X.N.

```
MACRO
BETWEEN X.1-X.N LOW HI ANS;
  STRICT.
MCOLUMN X.1-X.N L H ANS
MFREE     LOW HI
#
# X.1-X.N is a list of columns. LOW and HI 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 HI. 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 >=.
#
RMINIMUM X.1-X.N L
RMAXIMUN X.1-X.N H
# Case where subcommand is not used
IF STRICT = 0
  LET ANS = ( L >= LOW ) AND ( H <= HI )
# Case where subcommand is used
ELSE
  LET ANS = ( L > LOW ) AND ( H < HI )
ENDIF
ENDMACRO
```

You can invoke BETWEEN in any of the following ways:

%BETWEEN C1-C3 .25 .35 C10

%BETWEEN C1-C3 C4 .35 C10

%BETWEEN C1-C3 .25 C5 C10

%BETWEEN C1-C3 C4 C5 C10

You can write a macro where a suffixed list of variables is declared as MFREE. But recall that all variables in a suffixed list must be of one type. Thus, in any one invocation of this macro, all the variables in the list must be of the same type. If you need to know what type of variable was passed in, use MTYPE.

MTITLE: Session command for adding a title above output

MTITLE ***"title"***

Starts the `MTITLE` mode. This mode adds a title for any output that is produced by commands between `MTITLE` and `ENDMTITLE`. While in `MTITLE` mode, you cannot save a project, open a project, open a worksheet, or open a graph.

NOTITLE

Suppresses the titles of the commands executed while in `MTITLE` mode.

ENDMTITLE

Ends the `MTITLE` mode. Minitab does not display any output until you specify `ENDMTITLE`.

NEXT: Session command for transferring control from a loop to the beginning of the block

Transfers control from within a DO- or WHILE-loop back to the beginning of the block. For DO, the loop variable is then set to the next value in the list and the loop is executed again. The following is a simple example, using a global macro.

```
GMACRO
FIVES
#
# Takes the column named X and changes all entries
# that are greater than 5 to 5.
# Constants K90 and K91 are used for scratch work.
#
NAME K90 'N' K91 'I'
LET 'N' = COUNT('X')
DO 'I' = 1 : 'N'
  IF  'X'('I') <= 5
    NEXT
  ELSE
    LET 'X'('I') = 5
  ENDIF
ENDDO
ENDMACRO
```

The DO-loop goes through all the values in X. If a value is less than or equal to 5, NEXT passes control to the top of the DO-loop and the value is left unchanged. If a value is greater than 5, the ELSEIF block is executed and that value is set to 5.

NOBRUSH: Session subcommand for disabling brushing on a graph

Can be used as a subcommand of any graphics command to disable brushing on the resulting graph. Why disable brushing? Brushing can only highlight rows of data in the global worksheet. But graphs created in local macros are sometimes based on data in the local worksheet that have no relationship to corresponding rows of data in the global worksheet.

NOTE: Session command for adding comments that are displayed in the output

`NOTE`

Use NOTE to annotate your macro program with comments that are displayed in the output. To annotate with comments that are not displayed, use the comment symbol #.

Use notes to make your output more readable with spaces or to add helpful notes to yourself. Text from notes will appear in separate output tabs unless you use the `MTITLE` and `ENDMTITLE` commands to specify a block of output.

- Put the `NOTE` command at the beginning of a line.
- All text on that line will be ignored by the macro processor. However, text on a `NOTE` line (except the first five spaces – the word `NOTE` and a space) does display in the output when the macro is executed.

Note The `NOTE` cannot end with a semicolon. You can use a semicolon, but end the line with it. For example, NOTE AB;CD is acceptable, but NOTE ABCD; is not acceptable.

You can also make your macro file more readable by adding blank lines between the lines of macro statements and commands. The blank lines do not interfere with the execution of the macro, and will not appear in the output. You do not have to start a blank line with the comment symbol #.

Example of using NOTE in a macro

Macro Code

```
GMACRO
SAMPLEMACRO
RAND 50 C1 C2 C3;
UNIFORM 5 10.
MTITLE "Print Columns".
NOTE Here come the data
NOTE
PRINT C1-C3.
ENDMTITLE
ENDMACRO
```

Results

```
Here come the data

Data
 Row   Yield   Chem1   Chem2
   1   11.28     87     1.83
   2    8.44     61    25.42
   3   13.19     59    28.64
...
```

Note If you don't use `MTITLE`and `ENDMTITLE`, then this example will generate 2 tabs of output. The instance of NOTE without any text does not generate an output tab.

1. The first tab will contain Here come the data.
2. The second tab will be blank.
3. The third tab will contain the data.

PAUSE and RESUME: Session commands for pausing and resuming a macro

`PAUSE`

When Minitab encounters a PAUSE in a macro, control is shifted from the macro to the keyboard. You can then type any Minitab command. PAUSE can help you debug a macro you are developing. It can also allow you to get input from the macro user.

If you are in PAUSE mode from within a local macro, you have access to the local worksheet and only the local worksheet. You can also declare new local variables and use them. They will be stored at the end of the local worksheet.

When you are in PAUSE mode, you can type any Minitab command. You cannot CALL other macros in the same file, invoke a macro from another macro file, or use control statements.

`RESUME`

When you want to return control to the macro, type RESUME (or just R).

READ, TSET, and SET: Session command for asking users questions and using the answers in a macro

READ, TSET and SET have a special feature that allows you to ask users questions and then use their answers in the macro. A macro will pause for user input if you use READ, TSET or SET with the subcommand FILE with the special filename TERMINAL. TERMINAL tells Minitab to wait for input from the keyboard.

READ, TSET and SET have other subcommands.

```
READ C...C

    FILE "TERMINAL"

SET C

    FILE "TERMINAL"

TSET C

    FILE "TERMINAL"
```

RETRIEVE: Session command for retrieving a saved worksheet or project

```
RETRIEVE "filename"

RETRIEVE K
```

Note The menu command **File** > **Open** and the session command WOPEN also open Minitab saved worksheets and Excel files (and many other types of files). They provide several useful options that are not available with RETRIEVE.

Use the main command by itself to retrieve a saved worksheet and add the file to the current project. With subcommands, you can open a project or add one or more worksheets from a project to the current project. You can specify the filename as either the name of the file in double quotes or as a stored text constant.

If you omit the file name and the current folder contains a file named Minitab.MWX or Minitab.MTW, then Minitab opens that file.

Note You cannot use RETRIEVE in a local macro. For more information, go to Commands and subcommands that are not allowed in macros on page 29.

PROJECT

Note You cannot use `PROJECT` in a global macro. For more information, go to Commands and subcommands that are not allowed in macros on page 29.

Specifies that the file after `RETRIEVE` is a Minitab project file (MPX, MPJ). If you do not use `MERGE`, then Minitab asks whether you want to save the current project and opens the project that you specify with `RETRIEVE`. If you do not want the prompt, use the `SAVE` command with the `PROJECT` subcommand before you use the `RETRIEVE` command.

MERGE ***"worksheet name"..."worksheet name"***

Opens only the worksheet(s) that you specify from a project that you specify with `RETRIEVE`. The subcommand adds the worksheets to the current project. You can use `MERGE` only after you use `PROJECT`.

PASS ***"password"***

To retrieve a password-protected file, specify the password.

WHILE and ENDWHILE: Session commands for repeating a block of commands depending on a logical expression

WHILE ***logical expression***

Repeats a block of commands as long as a logical expression is true.

ENDWHILE

Marks the end of the WHILE loop.

Repeats a block of commands as long as the logical expression is true. The logical expression follows the same rules as in the IF statement.

Suppose you want to find the root of the equation, $y = -1 + x + x^3$. This equation has only one real root, which is between 0 and 1. The following global macro calculates, approximately, what the root is.

```
GMACRO
ROOT
#
# Finds the root of a specific polynomial. The result is
# within .01 of the exact answer.
# K90-K93 are used for scratch work
#
NAME  K90 'X'  K91 'Y'  K92 'Xlow'  K93 'Ylow'
LET 'X' = 0
LET 'Y' = -1
WHILE 'Y' < 0
  LET 'X' = 'X' + .01
  LET 'Y' = -1 + 'X' + 'X'**3
ENDWHILE
LET 'Xlow' = 'X' - .01
LET 'Ylow' = -1 + 'Xlow' + 'Xlow'**3
```

```
PRINT 'Xlow' 'Ylow'  'X' 'Y'
ENDMACRO
```

The macro first initializes the two variables, X and Y, to 0 and −1. Each time through the WHILE-loop, Minitab first checks to see that Y is still less than zero. If it is, Minitab increases X by .01 and calculates Y at this new value. When the condition fails—that is, when Y is no longer less than zero—the macro exits the loop and goes to the first statement after ENDWHILE. Then, the macro prints the result.

Note Instead of modifying a worksheet variable inside a WHILE / ENDWHILE loop, copying the worksheet variable to a local macro variable, modifying the macro variable in the loop, then copying the macro variable back to the worksheet variable might be faster.

WRITE: Session command for storing data in a text file

WRITE *E...E*

Writes data in the specified columns or constants to the screen or to a data file.

Because of potential conflicts with the global worksheet, the commands SAVE and RETRIEVE do not work in a local macro. Global worksheet variables that have been passed into the macro as arguments assume any new values given to them during the course of the macro execution. You can always save those variables after the macro executes. But you may also want to save local worksheet variables that are not passed as arguments. You can use WRITE to save local worksheet variables, use the WRITE command within your macro.

FILE *"filename"*

Specifies the file to store the data in.

Example

Suppose you have three column variables in the local worksheet named X, Y, and Z. The following command saves those three columns in a text file named MYWORK.DAT.

```
WRITE X Y Z;
  FILE "MYWORK".
```

WTITLE: Session subcommand for specifying the title of the output pane

WTITLE *"title"*

Specifies the title for the output pane.

You can use WTITLE as a subcommand with LAYOUT and all graph commands. The title you specify becomes the title of the output pane.